Service Layer Components for Decentralized Applications

Doctoral Thesis
(Dissertation)

to be awarded the degree of
Doctor rerum naturalium (Dr. rer. nat.)

submitted by
Fabian Stäber
born in Düsseldorf

approved by the Faculty of
Mathematics/Computer Sciences and Engineering,
Clausthal University of Technology

conducted at
Siemens Corporate Technology, Munich

Date of oral examination
7 November 2008

Chairperson of the Board of Examiners
Prof. Dr. Jürgen Dix

Chief Reviewer
Prof. Dr. Jörg P. Müller

Reviewer
Prof. Dr. Bernd Freisleben

Abstract

While the concept of decentralization in distributed systems is relatively old, it has been drawing increased attention since the rise of peer-to-peer systems in 2000. A significant number of research results has been published, enabling self-organization, scalability, and resilience. However, the adoption of these results in industry is still limited. One reason is that there is no methodology available that helps industrial application developers to transfer these results to their domains.

The objectives of this thesis are twofold: Firstly, an architecture for decentralized applications is introduced. Based on this architecture, a methodology is presented that supports application developers in benefiting from state-of-the-art decentralization in their fields of application, and in identifying requirements that have not yet been addressed in related work. Secondly, the architecture and methodology are applied to three industrial application scenarios; for each of these scenarios, the open requirements are identified and new service components are developed, extending the state of the art and enabling the use of decentralized infrastructures.

The architecture is based on the analysis of the following three industrial application scenarios:

1. A decentralized business collaboration platform for the automotive industry, enabling collaborative business processes for a continuously changing set of business partners.

2. A user directory for a decentralized communication platform supporting, for example, video communication, voice communication, and instant messaging.

3. A control infrastructure for distributed power generation that provides reliable routing of sensor and control data even in the potentially unreliable Internet.

The key outcome of the analysis is that most requirements appear in more than one scenario in a similar way, in spite of the diversity of the application domains. Therefore, the architecture presented in this work introduces

a component-based service layer, providing functionality as generic, domain-independent building blocks. Current research on peer-to-peer-based decentralization is reviewed, and functionality is generalized as reusable service layer components. The methodology presented in this thesis relies on relating these components to the application requirements. That way, developers can identify gaps between the state of the art and the application requirements, and focus on the problems that have not yet been addressed in related work.

The methodology is applied to each of the three afore mentioned application scenarios. The gap between the application requirements and what state-of-the-art decentralization technologies offer is identified. New components are developed to close that gap. That way, peer-to-peer-based decentralization becomes applicable more easily in these scenarios.

The evaluation of the three new service layer components shows that they fulfill the application requirements. The evaluation of the architecture and the methodology shows that the approach can be generalized and applied to all three scenarios in a similar way. Therefore, it is plausible that the architecture and methodology presented in this thesis can be applied to future scenarios as well, helping industrial application developers to use peer-to-peer-based decentralization in their fields of applications. The thesis concludes with a discussion of limitations and future research directions.

Zusammenfassung

Das Konzept der Dezentralisierung ist mit dem Erfolg von Peer-to-Peer Systemen seit dem Jahr 2000 wieder mehr ins Zentrum der Aufmerksamkeit gerückt. Eine Vielzahl an Forschungsergebnissen wurde erzielt, die Selbstorganisation, Skalierbarkeit und Zuverlässigkeit ermöglichen. Trotzdem ist die Verbreitung dieser Ergebnisse in der Industrie immer noch beschränkt. Der Grund dafür ist, dass keine Methodik zur Verfügung steht, die es Entwicklern industrieller Anwendungen ermöglicht, Forschungsergebnisse in ihre Anwendungsbereiche zu übertragen.

Die vorliegende Arbeit verfolgt zwei Ziele: Erstens wird eine Architektur für dezentrale Anwendungen entworfen. Von dieser Architektur wird eine Methodik abgeleitet, die es Entwicklern industrieller Anwendungen ermöglicht, neue Ergebnisse im Bereich der Dezentralisierung in ihre Anwendungsbereiche zu übertragen und bisher ungelöste Anforderungen zu identifizieren. Zweitens werden diese Architektur und Methodik auf drei industrielle Szenarien angewendet; für jedes dieser Szenarien werden die ungelösten Anforderungen identifiziert und es werden neue Service-Komponenten entwickelt, die den Stand der Forschung erweitert und den Einsatz dezentraler Infrastrukturen ermöglicht.

Der Architektur liegt die Analyse der folgenden drei industriellen Anwendungsszenarien zugrunde:

- Eine dezentrale Plattform für Geschäftspartner in der Automobilindustrie, die die Abwicklung firmenübergreifender Geschäftsprozesse ermöglicht.
- Ein Nutzerverzeichnis für eine dezentrale Kommunikationsplattform, die beispielsweise Bildkommunikation, Sprachkommunikation und den Austausch von Textnachrichten unterstützt.
- Eine Kontrollinfrastruktur für verteilte Energieerzeugung, die zuverlässiges Routing von Sensor- und Kontrolldaten ermöglicht, auch im potentiell unzuverlässigen Internet.

Die Analyse zeigt, dass die meisten Anforderungen in mehreren Szenarien auf ähnliche Weise wiederkehren, obwohl die einzelnen Szenarien aus sehr

unterschiedlichen Anwendungsfeldern gewählt wurden. Deshalb wird in dieser Arbeit eine Architektur vorgestellt, die einen komponentenbasierten Service Layer einführt, der Funktionalität als generische, anwendungsunabhängige Komponenten zur Verfügung stellt. Aktuelle Forschungsarbeiten im Bereich Dezentralisierung werden betrachtet und deren Funktionalität als wiederverwendbare Service-Komponenten generalisiert. Die Methodik, die in dieser Arbeit vorgestellt wird, beruht darauf, diese Komponenten mit den Anwendungsanforderungen in Relation zu setzen. Dadurch können Entwickler die Lücke zwischen dem Stand der Forschung und ihren Anwendungsanforderungen identifizieren und sich auf jene Anforderungen konzentrieren, die noch nicht in anderen Arbeiten behandelt wurden.

Diese Methodik wird auf die drei oben genannten Szenarien angewendet, die Lücke zwischen dem Stand der Forschung und den Anforderungen der Anwendungen wird identifiziert, und neue Lösungen werden entwickelt, die diese Lücke schließen. Dadurch wird die Verwendung Peer-to-Peer-basierter Dezentralisierung in diesen Szenarien ermöglicht.

Die Evaluierung der drei neuen Service-Layer-Komponenten zeigt, dass die Anforderungen der Anwendung erfüllt werden. Die Evaluierung der Architektur und der Methodik zeigt, dass der Ansatz verallgemeinert und in allen drei Szenarien auf ähnliche Weise angewendet werden kann. Daher ist es plausibel, dass die Architektur und die Methodik auch auf zukünftige Szenarien übertragbar sind, um Entwicklern industrieller Anwendungen helfen, Peer-to-Peer-basierte Dezentralisierung in ihren Anwendungen zu nutzen. Die Arbeit endet mit einer Betrachtung der Grenzen des vorgestellten Ansatzes und mit einem Ausblick auf zukünftige Entwicklungen.

Acknowledgements

This work would not have been possible without the help of many people. I would like to thank my supervisors at university, Prof. Dr. Jörg P. Müller and Prof. Dr. Bernd Freisleben, as well as my supervisor at Siemens, Dr. Gerd Völksen, for their valuable feedback.

Furthermore, I would like to thank the current and former members of the peer-to-peer center of competence at Siemens Corporate Technology for their support and enlightening discussions, namely Alan Southall, Steffen Rusitschka, Sebastian Dippl, Kolja Eger, Dr. Thomas Friese, Christoph Gerdes, Christian Kleegrewe, Stephan Merk, Dr. Dieter Olpp, and Sebnem Öztunali.

My thanks also go to the former department head Rudolf Kober and his successor Dr. Burghard Schallenberger for supporting my publications and conference participations.

Many thanks to my fellow Udo Bartlang for the good cooperation and the helpful discussions.

Contents

1. Introduction

1.1. Decentralization and Peer-to-Peer Computing

The goal of decentralized distributed systems is to implement applications as collaborative tasks, without using centralized servers or coordinators. The absence of centralized services enables decentralized applications to provide a high level of scalability and reliability.

The appearance of file sharing communities in the year 2000 initiated a major change in the Internet [70]. While former Internet applications tended to use classical client-server architectures, the file sharing communities introduced the new peer-to-peer paradigm.

In a classical two-tier client-server architecture, a centralized server provides resources, e.g. Web pages, and the clients request these resources on demand. All resources are stored on the server, the storage space of the clients remains unused.

As computers continuously became more powerful and less expensive, a lot of unused storage and CPU power accumulated on the clients at the edge of the Internet. The file sharing applications were the first applications to exploit this potential.

The file sharing applications introduced the peer-to-peer paradigm as an alternative to the classical client-server-based architectures. The idea is that each peer is a client and a server at the same time. Each peer may use resources exposed by other peers and vice versa. As a result, a community is created, providing access to virtual storage space and CPU power that is physically distributed among the peers in a peer-to-peer overlay.

Napster[1], as the first commercially successful file sharing application, used a centralized index server to coordinate the peers. However, the peer-to-peer paradigm was soon adopted by the research community and became a new branch in decentralization, complementing other technologies, e.g. multi-agent systems. In 2001, several new protocols for peer-to-peer overlays were published, implementing fully decentralized peer-to-peer approaches [106, 84, 117, 91].

Section 2 gives an overview of current developments in decentralization

[1] http://www.napster.com, 19 November 2007

and introduces state-of-the-art decentralized peer-to-peer overlays. A detailed analysis of the most influential protocols is provided in Appendix A.

1.2. Goal of this Thesis: Developing and Applying a Methodology for Adopting Decentralization in Industrial Applications

Efficient use of existing resources, high scalability, resilience, and self-organization of decentralized peer-to-peer overlays make the peer-to-peer paradigm promising for a wide range of applications. However, the impact of peer-to-peer-based decentralization on industry is still very limited, with Voice over IP telephony (VoIP) as a rare exception.

The reason for this is that state-of-the-art peer-to-peer research is very hard to utilize in new application domains. Many peer-to-peer systems are monolithic applications, being designed with a specific use case in mind. For example, peer-to-peer-based file sharing applications provide CRUD[2]-like interfaces, hiding complex mechanisms like replication [117], access control [93], load balancing [19], data consistency [86], etc.

The contribution of this thesis is twofold:

- An architecture and methodology are presented that help developers from industry to transfer current research results in peer-to-peer-based decentralization to their fields of application, and to identify the gap between the state of the art and the application requirements.
- The methodology is applied to three industrial application scenarios. For each of these scenarios a new component is developed closing the respective gap. These components enable the implementation of peer-to-peer-based decentralization in new areas of application.

The following two subsections summarize these contributions.

1.2.1. Architecture and Methodology

Current peer-to-peer systems aim at specific use case scenarios, e.g. filesharing, and include a lot of domain-specific functionality. This results in large, monolithic applications which makes it difficult for application developers from industry to transfer state-of-the-art peer-to-peer computing to new application domains.

[2] CRUD: Create, read, update and delete.

There is no generic architecture for building customized decentralized infrastructures, and there is no methodology that can be used to transfer current research results in decentralization to new fields of application.

In this thesis, an architecture for peer-to-peer-based decentralized applications is presented. The architecture is derived from an analysis of three industrial application scenarios. The architecture proposes a lightweight overlay layer taking the pure routing and aggregation functionality. The overlay is complemented by a component-based service layer, providing application-specific functionality. The service layer is built of generic components, each of which providing a certain feature, e.g. replication, caching, atomic transactions, etc.

From this architecture, a methodology is derived that helps application developers to transfer state-of-the-art research results to their fields of application. The idea is to relate functionality provided by service layer components to the application requirements. As a result, missing functionality is found and the requirements for additional service layer components are identified. The developer focuses on implementing these additional components, and integrates these new components into the overall application.

As the architecture and methodology are applied to three different application scenarios in a similar way, conclusions can be drawn that show which part of the approach is use case specific, and which results are similar in all of the use cases.

1.2.2. New Service Layer Components

The architecture and methodology are applied to three application scenarios from industry. For each of these scenarios, the gap between state-of-the-art functionality and the application requirements is identified. The gap analysis yields the requirements for novel components. As part of this thesis, three new components are developed that enable peer-to-peer-based decentralization in the respective application scenarios:

Confidential Communication. The gap analysis for a business collaboration platform shows that related work does not provide functionality for implementing confidential communication in structured peer-to-peer overlays. Therefore, a confidential communication component needs to be developed. The component is based on onion routing, which is an algorithm that has originally been developed for anonymous email. The idea behind this is transferred to a decentralized collaboration platform, and peer-to-peer-specific functionality is added to comply with the security

requirements[3].

Search Index. The search index component enables range queries in a distributed user directory. Although a lot of related work on range queries in peer-to-peer overlays is available, the evaluation shows that none of these approaches complies with the scalability requirements when applied to Zipf-distributed user entries in a distributed user directory. Therefore, a new search index is developed that provides a tree structure that can be optimized for user entries. The search provides several optimization features, and it can be combined with a caching component to provide high scalability. The differences between the search index component and other approaches are shown in Section 6.

Routing and Aggregation. The routing and aggregation component implements a scalable and reliable multicast infrastructure for control and sensor data in distributed power generation. As a result, peer-to-peer infrastructures become available for power generation scenarios, and scalability for the increasing number of alternative power generation devices can be achieved.

These new components enable the use of peer-to-peer-based decentralization in the respective application scenario. That way, the applications benefit from the resilience, scalability, and self-organization provided by decentralized peer-to-peer overlays.

1.3. An Example of Current Approaches in Decentralized Applications

This subsection presents Tapestry [116] as an example of current decentralized infrastructures. The intention is to exemplify the difficulties arising when transferring current decentralization techniques to industrial applications, and to motivate the approach proposed in this thesis.

The architecture proposed in Section 3 and the application scenarios introduced in Section 4 are anticipated here in order to show the scope of this thesis. Tapestry serves as an example and similar observations could be made with other decentralized infrastructures as well.

Tapestry is a self-organizing, distributed infrastructure to be deployed as an overlay on top of a TCP/IP network. Tapestry is resilient to node failure, and

[3] For example, a pool of public keys and peer identifiers needs to be maintained in order to enable the silent creation of confidential communication paths.

provides a **decentralized object location and routing (DOLR)** interface [28], i.e. data objects can be queried using unique identifiers, and Tapestry resolves near-by peers providing these data objects. Tapestry's routing algorithm will be introduced in Appendix A. This subsection focuses on the applicability of Tapestry-like infrastructures in industrial applications.

Tapestry was designed as a generic platform, serving as the underlying infrastructure for the OceanStore [86] document storage utility, the Bayeux [118] multimedia streaming system, and other applications [116].

The architecture of Tapestry and other current overlay infrastructures differs from the architecture that will be introduced in this thesis. In Section 3, a service layer will be presented that is located between the routing infrastructure and the application layer. The service layer complements the raw routing functionality of the overlay infrastructure. In contrast, Tapestry-based applications are built directly on top of the overlay. In order to provide better support for application developers, Tapestry integrates a lot of functionality directly into the overlay infrastructure. In particular, Tapestry natively supports many requirements found in the OceanStore application.

This enables application developers to implement Tapestry-based applications efficiently, if the application requirements match Tapestry's functionality. However, the experiences made with the industrial application scenarios in this thesis show that the "monolithic" approach taken in Tapestry and related applications makes it difficult to transfer the results to new industrial application scenarios.

The following examples anticipate the three application scenarios that will be presented in Section 4. The goal is to exemplify the challenges arising when transferring Tapestry-like infrastructures to new fields of application.

Business Collaboration. Tapestry's distributed data store is motivated by the Napster[4] file sharing application. When a peer shares data in the overlay, a pointer to the peer's IP address and port number is published in the overlay infrastructure. The actual data is stored on the peer and must be downloaded from that peer using a classical client-server protocol. Tapestry is merely used for locating near-by copies of the data.

This approach meets the requirements of file sharing-like applications. However, if the last peer that keeps a copy of a document fails, the document is no longer available in Tapestry. Therefore, this approach cannot be applied when documents need to be stored persistently, as in the business collaboration scenario. Moreover, Tapestry allows arbitrary peers to provide copies of the same document. While this is reasonable

[4] `http://www.napster.com`, 9 January 2008

in file sharing-like scenarios, it makes data consistency and security more difficult.

User Directory. As Section 6 will show, the distributed user directory uses an application-specific caching strategy, where frequent prefixes are cached more often than infrequent prefixes. This caching strategy conflicts with Tapestry's built-in caching functionality.

Moreover, Tapestry introduces its own Public Key Infrastructure (PKI) for assigning identifiers to peers. The user directory is developed for a communication platform where peers are operated in a Virtual Private Network (VPN) providing its own PKI. It would be desirable to re-use that PKI within the overlay in order to minimize administrative overhead.

Distributed Power Generation. Tapestry's combination of replication and locality-awareness results in information to be replicated on peers that are physically close to each other. However, in the distributed power generation scenario it is likely that several peers are operated in the same computing center. Therefore, if the computing center becomes unavailable, a whole group of near-by peers fails at the same time. In that case, Tapestry's locality-awareness increases the probability of data loss.

The examples above show how built-in functionality in Tapestry conflicts with the requirements of the application scenarios. The applications could be implemented more easily if Tapestry was reduced to a raw decentralized routing infrastructure, and if additional functionality was made available as optional components to be combined with respect to the application-specific requirements.

This thesis proposes a component-based service layer, fulfilling application requirements on top of a raw overlay infrastructure. The service layer does not provide a standard interface, but the interface is assembled using the components that are necessary for a specific application. Thus, the service layer provides customized, application-specific interfaces, reducing the complexity on the application layer.

A methodology will be presented helping application developers to identify the gap between state-of-the-art components and application requirements. That way, developers can identify requirements for new components and focus on yet unsolved challenges. In Section 6 a new component is presented for each of the three application scenarios above, enabling the use of peer-to-peer in new fields of application.

1.4. Ways of Developing a Methodology

The subsections above present and exemplify the objectives of this thesis. This subsection establishes the approach that is taken to reach these objectives.

1.4.1. Overview of Scientific Approaches

In philosophy of science, two major scientific approaches are distinguished:

1. The **rationalist** or **constructivistic** approach accepts theoretical reasoning as the only source of knowledge. The goal is to establish a "coherent, consistent, and wide-ranging theoretical organization" [18, 383b, 388c]. Empirical facts are merely examples illustrating the theory.

   With respect to this thesis, this would mean to investigate peer-to-peer computing from a theoretical point of view, and establish an abstract, generic architecture and methodology for the design of peer-to-peer-based applications. The concrete use cases would merely provide examples for the theoretical framework.

2. The **empiricistic** approach is based on real-life observations as the foundation of research. The goal is to infer novel predictions from generalizing empirical observations, and to verify these generalizations by comparing them with empirical facts [18, 388a].

   When applied to this thesis, this would mean to derive an architecture and methodology from the concrete use cases, and to try to generalize the results.

In their pure form, both approaches raise difficulties. The problem with the first approach is that although the resulting theory may be consistent on its own, the result might have little value for solving real-life problems.

The second approach will certainly result in a theory being helpful with real-life challenges, as it is based on the analysis of actual use cases. However, the resulting theory might be limited to a small set of applications, as only a limited set of observations has been considered in the first place.

1.4.2. Approach Taken in this Thesis

In this thesis, practical relevance of the results is one of the main objectives. Therefore, this thesis is based on the analysis of three use case scenarios. The architecture and methodology presented here have been derived from this analysis.

However, it is desirable that the results are applicable in a wide range of applications. The results should not be limited to a small set of use case scenarios. Therefore, the case studies are chosen from three very different application domains, each of which having a distinct focus.

The contribution of this thesis is twofold: Firstly, an architecture and methodology for designing peer-to-peer-based applications is presented. Secondly, three new software components are presented, enabling the application of peer-to-peer in scenarios where peer-to-peer has not been applied before.

The use cases are chosen in a way that the open challenges are of a generic nature. The new components solve general problems, and can be applied in a similar way to other scenarios as well. The diversity of the application domains makes it possible to generalize the architecture and methodology, and to transfer the results to other application scenarios.

1.5. Outline

This thesis is organized as follows:

- Section 2: Background and Scope in Distributed Systems defines the scope of this work with respect to other research areas in distributed systems, and shows why the focus of this thesis is on decentralized peer-to-peer computing.
- Section 3: An Architecture for Peer-to-Peer-Based Decentralized Applications presents an architecture that defines a service layer being composed of generic, domain independent service layer components.
- Section 4: Application Scenarios introduces three application scenarios that could benefit from the self-organization and resilience provided by decentralized systems. For each of these applications, the requirements are analyzed.
- Section 5: Definition and Classification of Service Layer Components Implementing State-of-the-Art Algorithms reviews state-of-the-art in peer-to-peer-based decentralization and derives software components that can be used in the service layer introduced in Section 3.
- Section 6: Applying a Methodology for Deriving New Components analyzes the gap between state-of-the-art and the application requirements identified in Section 4. New components are presented closing that gap.
- Section 7: Evaluation presents the evaluation of the methodology and the new components.

- Section 8: Conclusions & Future Work presents the conclusions and outlook.
- Appendix A: Current DHT Implementations gives an overview of the most important decentralized peer-to-peer protocols.

1.6. List of Publications

The following papers have been published as part of the research conducted in the context of this thesis:

1. Klaus Fischer, Jörg P. Müller, Fabian Stäber, and Thomas Friese. Using peer-to-peer protocols to enable implicit communication in a BDI agent architecture. In *ProMAS'06: Proc. of the 4th International Workshop on Programming Multi-Agent Systems,* volume 4411 of *Lecture Notes in Artificial Intelligence,* pages 15--37, Springer, 2006.
2. Fabian Stäber, Udo Bartlang, and Jörg P. Müller. Using onion routing to secure peer-to-peer supported business collaboration. In *Proc. of eChallenges 2006,* volume 3 of *Exploiting the Knowledge Economy: Issues, Applications and Case Studies,* pages 181--188, IOS Press, 2006.
   (best paper award)
3. Fabian Stäber, Giorgio Sobrito, Jörg P. Müller, Udo Bartlang, and Thomas Friese. Interoperability challenges and solutions in automotive collaborative product development. In *I-ESA'07: Proc. of the 3rd International Conference on Interoperability for Enterprise Software and Applications,* volume 2 of *Enterprise Interoperability,* pages 709--720, Springer, 2007.
4. Fabian Stäber and Jörg P. Müller. Evaluating peer-to-peer for loosely coupled business collaboration: A case study. In *BPM'07: Proc. of the 5th International Conference on Business Process Management,* volume 4714 of *Lecture Notes of Computer Science*, pages 141--148, Springer, 2007.
   (short paper)
5. Udo Bartlang, Fabian Stäber, and Jörg P. Müller. Introducing a JSR 170 standard compliant peer-to-peer content repository to support business collaboration. In *Proc. of eChallenges 2007,* volume 4 of *Exploiting the Knowledge Economy: Issues, Applications and Case Studies,* pages 814--821, IOS Press, 2007.
6. Fabian Stäber, Gerald Kunzmann, and Jörg P. Müller. Extended prefix hash trees for a distributed phone book application. In *ICPADS'07: Proc. of the*

17th International Conference on Parallel and Distributed Systems, IEEE Computer Society Press, 2007.
Also in *IJGHPC: International Journal of Grid and High Performance Computing*, to appear.

7. Fabian Stäber, Christoph Gerdes, and Jörg P. Müller. A peer-to-peer-based service infrastructure for distributed power generation. In *IFAC WC'08: Proc. of the 13th International Federation of Automatic Control World Congress,* 2008. *(to appear)*

8. Gerald Kunzmann, Andread Binzenhöfer, Fabian Stäber. Structured Overlay Networks as an enabler for future Internet services. In *it - information technology,* 50(6):376--381, 2008.

Part of this thesis is covered by the international patent PCT/EP2007/005611: "Verfahren zum Weiterleiten von Daten in einem dezentralen Datennetz".

2. Background and Scope in Distributed Systems

2.1. Introduction

This section defines the scope of this thesis within distributed systems research. It motivates why the focus of this thesis is on peer-to-peer-based decentralization, and introduces decentralized peer-to-peer computing in greater detail.

2.2. Related Research Areas

2.2.1. Overview

In this thesis, the term **distributed system** refers to all areas of computer science where two or more computers communicate with each other over a network. This subsection reviews related research topics in the field of distributed systems, and defines the scope of this work within that area.

The title of this thesis suggests which related topics are to be considered. Each of the terms in "service layer components for decentralized applications" is linked to some field in distributed systems research:

Service orientation has become a widely used concept in distributed systems, especially in business process IT. The paragraph on "Service Oriented Architecture (SOA)" shows how the work in this thesis is related with SOA, and where SOA services distinguish from the service layer components described in this thesis.

Layer decomposition is a very basic concept in system architectures. As part of this thesis, a layered architecture for decentralized applications is introduced. A component-based approach complements the layered architecture. The paragraph on "Layered Architectures in Distributed Systems" below shows the importance of layers in system architectures.

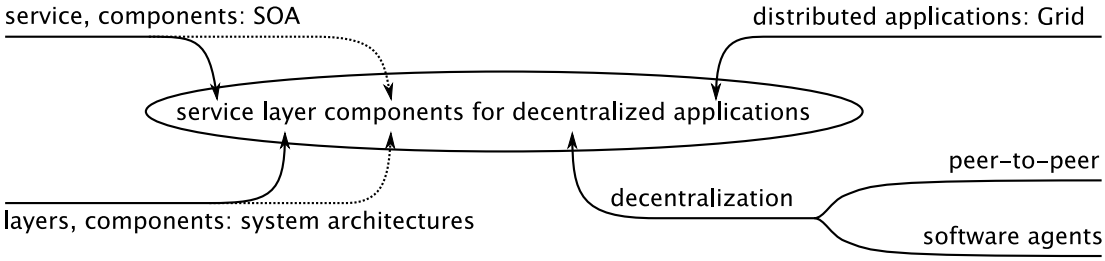


Figure 2.1.: Mind Map Showing Related Fields in Distributed Systems

Components play an important role in system architectures, especially in SOA. A discussion on component-based architectures is included in the paragraph on SOA.

Decentralized distributed systems have been a research topic for a long time. As a prominent example, multi-agent systems aim at implementing autonomous software agents solving problems in a collaborative way. Recently, decentralized peer-to-peer computing appeared as a new topic in decentralized distributed systems.

The paragraph on "Decentralization Using Agent Technologies" introduces agent technologies as a branch of decentralized distributed systems, and motivates why the focus in this thesis is on decentralized peer-to-peer computing.

Applications in distributed systems are targeted by Grid computing, which is a large research area aiming at distributed resource sharing. The Grid architecture combines layered architectures and service oriented approaches. The paragraph on "Distributed Applications in Grid Computing" below shows the differences between the technologies used in this thesis and related work from the Grid area.

The paragraphs below present these related fields, starting with layered architectures that have been used since the early days of the Internet.

2.2.2. Layered Architectures in Distributed Systems

System architectures in general all have one primary goal: Reducing complexity of the engineering process. In all areas of computer science this goal is targeted with the same basic concept: Abstraction [108].

This concept can be found in programming languages, where low-level programming languages are abstracted by high-level languages. It applies to the

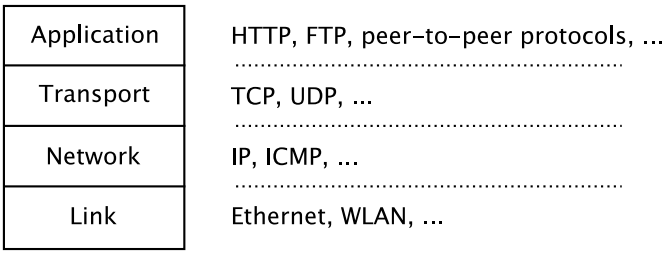


Figure 2.2.: The TCP/IP Reference Model

design of software applications, where abstraction is achieved with programming paradigms like object orientation, and it applies to all areas of distributed systems.

A widely used concept to implement abstraction in distributed systems is the use of **layers**. Each layer defines an interface exposed to the upper layer, and abstracts the details of the lower layer [56]. Figure 2.2 shows the TCP/IP reference model, which is one of the most prominent examples of a successful application of the concept of layers [108, p. 26].

In this thesis, a layered architecture for decentralized applications will be introduced and evaluated. A service layer will be introduced as a new level of abstraction, reducing the complexity of decentralized applications.

This layered architecture will be complemented by a component-based approach, which increases the level of abstraction, and provides additional flexibility for application architects. The overall architecture will be presented in Section 3.

2.2.3. Service Oriented Architecture (SOA)

While the idea of layers is comparably old, service-oriented design is a new development towards abstraction in distributed systems. The **service-oriented architecture (SOA)** is based on the idea that complex processes can be decomposed into activities that are plugged together to implement a process [21].

Today, SOA ranges among the most influential IT-trends in industry, especially in the area of business process IT. In order to benefit from the value of SOA, businesses undergo fundamental organizational and cultural changes [14]. As SOA is so widely known, the term **service** is often linked with the SOA architecture and its applications.

In this paragraph, the definition of SOA is reviewed, and contrasted with the concept of service layer components as presented in this thesis. The goal is to clarify the differences between SOA and the component-based architec-

ture presented in this thesis, and to show how component-based peer-to-peer infrastructures can be integrated into SOA applications.

OASIS Definition of Service

The OASIS organization defines a service as a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description [25].

As described in the Open Group's definition of SOA [45], a service may be composed of other services. However, to the consumers of a service, the service is a self-contained black box.

SOA is often associated with business process management, and therefore a service is often associated with an activity in a business process. However, the OASIS definition does not limit the service to providing business functionality. There can also be services providing access to technical capabilities.

Examples for Services

This thesis deals with the technical capabilities provided by different decentralized applications. A single peer-to-peer application does often provide more than one capability, i.e. in terms of SOA, a peer-to-peer application may provide several services.

The following list shows typical examples of the capabilities offered by current peer-to-peer applications:

- Peer-to-peer-based file sharing applications like BitTorrent-based applications typically provide functionality to publish content and to download content from other peers [23]. These operations would be the services in SOA terminology.
- Peer-to-peer-based multicasting infrastructures like PPLive [48] provide means for subscribing TV channels. The operations "subscribe" and "unsubscribe" are examples for services in a SOA.
- Peer-to-peer-based distributed computing infrastructures like SETI@home[1] [6] enable the distribution of computational tasks in order to perform costly computations in parallel. The services provided by these infrastructures include the deployment of tasks, the execution of the tasks, and the aggregation of the results. Related work on the integration of

[1] `http://setiathome.berkeley.edu`, 12 October 2007

distributed computing and service orientation includes the Open GRID Services Architecture (OGSA)[2] [38].

In all of the examples above, more than one service is provided by the respective peer-to-peer application.

Differences between Components in this Thesis and SOA Services

The architecture presented in this thesis takes the component-based approach of SOA, and transfers it to decentralized applications. In Section 3, a component-based architecture for peer-to-peer-based decentralized applications will be introduced as a replacement for the monolithic architectures used in current implementations.

The term "service layer component" will be introduced, denominating the building blocks that can be combined to implement peer-to-peer-based decentralized applications. Just like in SOA, some of the service layer components can be used in parallel, while others need to be involved sequentially, like in a workflow.

However, there are differences between SOA services and the service layer components as defined in this thesis. SOA services are considered as activities being executed locally by a certain entity providing that service. Service layer components implement distributed protocols in peer-to-peer-based applications. Thus, involving a service layer component usually triggers interaction among several peers in a peer-to-peer overlay.

The service layer components introduced in this thesis must be installed on all peers in the peer-to-peer overlay. There are no distinguished peers providing a certain functionality. All peers provide equal capabilities. Therefore, all service layer components can be triggered on the local peer. While SOA research has a strong focus on the interfaces making services available remotely, the service layer components introduced in this work can be accessed locally as a library.

To summarize the considerations above, the similarities between SOA and the present thesis is the component-based approach, defining an application as a composition of building blocks, each of which fulfilling certain application requirements. However, this thesis focuses on the functionality of these components and its implications in a decentralized environment, while SOA-related work focuses on interfaces.

[2] http://www.globus.org/ogsa, 12 October 2007

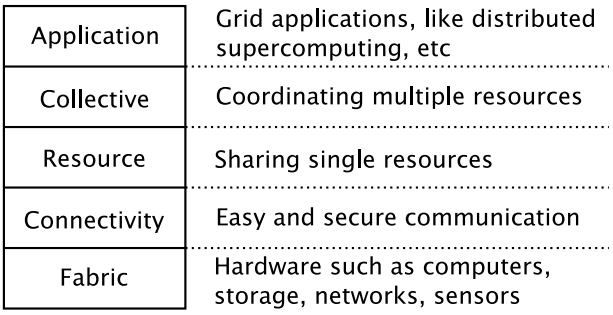


Figure 2.3.: The Anatomy of the Grid [39]

2.2.4. Distributed Applications in Grid Computing

The **Grid** is a new infrastructure that builds on today's Internet and Web to enable and exploit large-scale sharing of resources within distributed, often loosely coordinated groups [35].

The large community of Grid researchers, and the availability of the open source **Globus Toolkit** [36] have made Grid research an important branch of distributed computing.

Like peer-to-peer computing, Grid computing is concerned with enabling resource sharing within distributed collaborations. The collaborating groups are traditionally smaller and better connected than the groups looked at in peer-to-peer research, but in some regard the objectives of Grids and peer-to-peer are overlapping [37, p. 293].

The present thesis has its focus on peer-to-peer-based decentralization. This paragraph will show how the work in this thesis is related to Grid research.

Anatomy of the Grid

Figure 2.3 shows the layered architecture used in Grid applications [39]. This paragraph provides a short review of these layers, and shows where peer-to-peer is located with respect to the Grid anatomy.

On the lowest level, a *fabric* layer provides interfaces to control the resources to be shared in the Grid. On top of that, a *connectivity* layer enables the secure exchange of data between fabric layer resources. The *resource* layer uses these secure connections, and implements information and management protocols for the shared resources. The *collective* layer abstracts from specific resources, and provides virtual organizations. On top of that, the *application* layer implements the user applications that operate in the Grid [37, pp. 47--53].

Relation with Peer-to-Peer

Peer-to-peer computing provides abstract resources, e.g. a data repository that is physically distributed among all the peers forming a peer-to-peer overlay. Therefore, it seems natural that peer-to-peer computing is similar to the Grid's collective layer that builds virtual organizations from multiple resources.

A comparison of related work in peer-to-peer and Grid computing shows that both research areas target the same general problem. Although both approaches have their specific strength and weaknesses, the long-term objectives of the two communities seem likely to converge [40].

However, the present thesis does not aim at that direction. In this thesis, three application scenarios for peer-to-peer computing are addressed. Each of these applications would be a fabric layer component with respect to the Grid architecture.

To exemplify this, the following considerations show the implications if a peer-to-peer-based data repository was to be integrated in the Grid environment. Firstly, the security features of the connectivity layer would need to be implemented, enabling the integration of the repository with the Grid's PKI. Then, resource management and control would need to be added on top of the connectivity layer. Finally the repository would need to be integrated into a virtual organization.

As a result, all the Grid functionality from the connectivity layer to the top would have to be added. From the Grid's point of view, the data repository does not provide more functionality than a single hard disk, except that the data repository is larger and more reliable.

Of course, in order to use a data repository in the Grid environment, it is necessary that the repository provides interfaces enabling the implementation of access control, monitoring, etc, i.e. the semantics and requirements of the Grid layers must be known to the peer-to-peer developer.

However, this thesis looks at peer-to-peer-only applications that are not integrated in the Grid. Therefore, the Grid interfaces are not among the requirements addressed in this thesis.

2.2.5. Decentralization Using Agent Technologies

Decentralization has been a research topic for a long time. Autonomous agents are among the most wide-spread and mature technologies in this area.

Recently, peer-to-peer has become a new trend in decentralization. This paragraph shows the differences between peer-to-peer and agent technologies, and explains why this thesis has its focus on peer-to-peer computing.

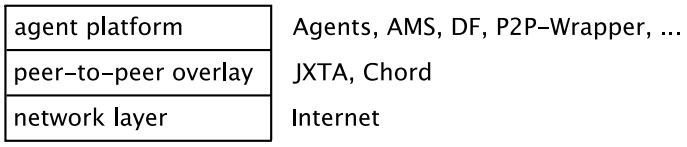


Figure 2.4.: Architecture for Integrating Peer-to-Peer and Agent Technologies

Differences between Agent Technologies and Peer-to-Peer Computing

In their weak notion of agency, Wooldridge and Jennings define four criteria that serve as the most general properties of an agent [115]:

Autonomy. Agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.

Social ability. Agents interact with other agents (and possibly humans) via some kind of agent-communication language.

Reactivity. Agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it.

Pro-activeness. Agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative.

Based on these definitions, the difference of agents and peers can be defined. The first three criteria apply to peers as well. Peers are autonomous, they communicate with each other using a peer-to-peer protocol, and they react on changes in the environment, i.e. they adapt their routing tables when other peers join or leave the peer-to-peer overlay.

However, the fourth property does not apply to peers, which shows the difference between agent technologies and peer-to-peer computing. While peer-to-peer computing simply maintains routing infrastructures, agent technologies implement decentralized applications.

Integration of Agent Technologies and Peer-to-Peer

The review of the agent definition above showed the differences in scope of agent technologies and peer-to-peer computing. However, related work shows how peer-to-peer and agent technologies can complement each other [44, 33].

In peer-to-peer-enabled agent systems, peer-to-peer is used as an underlying communication platform, and the agents are implemented on top of that platform. The resulting architecture is illustrated in Figure 2.4. That way, peer-to-peer can be used to fully decentralize the communication among the agents.

An example for that can be found in the FIPA[3] architecture [114]. An agent platform includes two facilities: An Agent Management System (AMS) that can be used as white pages to locate agents, and a Directory Facilitator (DF) that serves as yellow pages to look up services. Both of these facilities can be fully decentralized using a peer-to-peer overlay as the underlying communication infrastructure.

Decentralization in this Thesis

In the context of this thesis, decentralization refers to decentralized infrastructures and not to decentralized decision making. Therefore, with respect to the weak definition of agent technologies above, this thesis is more concerned with peer-to-peer than with agents.

2.2.6. Summary

Above, the scope of this thesis within distributed systems research was described. Related research topics were reviewed and differentiated from this work. In this thesis, decentralization is implemented using peer-to-peer computing. In the remainder of this section, the focus on peer-to-peer computing is motivated, and an overview of decentralization and peer-to-peer is given.

2.3. Decentralization Using Peer-to-Peer Computing

2.3.1. Overview

As described above, decentralization has been a research topic for a long time, e.g. in multi-agent systems [53]. However, research on decentralization has received increased attention with the appearance of peer-to-peer computing [62].

Peer-to-peer computing started to become popular with the success of file sharing platforms in summer 2000 [70]. The peer-to-peer paradigm was soon picked up by academic research, and the most influential publications on peer-to-peer overlay architectures were published in 2001 [84, 106, 91, 117].

[3] FIPA: Foundation for Intelligent Physical Agents

Providing resilient and self-organizing architectures, decentralized peer-to-peer overlays became an interesting new branch in research on decentralized distributed systems. However, although a substantial amount of research has been available for years, decentralized peer-to-peer systems do not yet play a significant role in decentralizing industrial applications. The reason for this is that there is no methodology that allows application developers to transfer research results in peer-to-peer computing to industrial engineering [102].

The scope of this thesis is not on the peer-to-peer algorithms themselves, but on a service layer being located on top of the peer-to-peer overlay. The goal is to define and evaluate an architecture and methodology that supports software architects to adopt peer-to-peer-based decentralization in new fields of application.

In the remainder of this section, state-of-the-art decentralized peer-to-peer algorithms are described. The goals are to give an overview of different overlay topologies, to motivate the focus on structured overlays, and to introduce a common terminology that is used in the remainder of this thesis.

The presentation below provides an abstract view of the peer-to-peer algorithms. The technical details of the most common protocols are described in Appendix A.

2.3.2. What is Peer-to-Peer Computing?

In related publications, the term peer-to-peer is used differently in different contexts. The following three meanings can be found:

- The term **peer-to-peer paradigm** describes a system paradigm with no specialized entities for certain tasks. All entities have the same abilities and operate in the same way. This can be found in technical scenarios [104] as well as in non-technical scenarios[4]. For example, in a review process for a conference, each attendant may be an author and a reviewer at the same time. Therefore, these processes implement the peer-to-peer paradigm.

- The term **peer-to-peer application** is used to denominate entire peer-to-peer-based applications. Typical examples are file sharing platforms, VoIP telephony, or SETI@home [6].

- The **peer-to-peer overlay** is the addressing scheme and routing protocol underlying a peer-to-peer application. The service layer presented in this thesis is implemented on top of the raw peer-to-peer overlay.

[4] `http://www.p2pfoundation.net` 11 January 2008

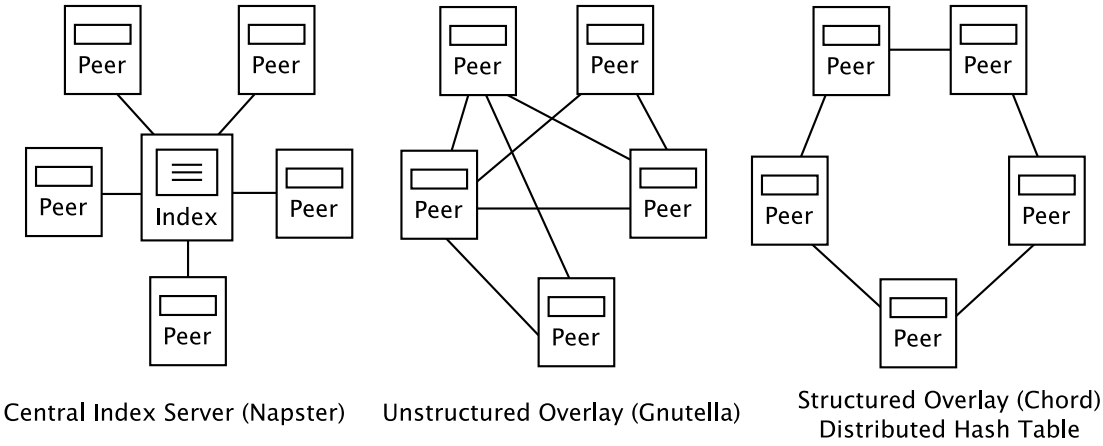


Figure 2.5.: Three Topologies for Peer-to-Peer Overlays.

The exact line between the raw peer-to-peer overlay and the peer-to-peer application will be introduced with the architecture in Section 3. The next paragraph introduces different types of peer-to-peer overlays and motivates the focus on structured overlays in this thesis.

2.3.3. Survey of Peer-to-Peer Overlay Topologies

Above, a peer-to-peer overlay was characterized as a distributed routing and addressing scheme. Using the overlay one can query a set of identifiers, and the query is routed to a set of peers being responsible for these identifiers.

In the following, different ways to implement an overlay are presented. This paragraph motivates why the focus of this work is on structured peer-to-peer overlays.

Peer-to-peer overlays can be divided into three different types, as illustrated in Figure 2.5. All of these types have in common the basic functionality provided by peer-to-peer overlays: A set of identifiers is mapped to a set of peers being responsible for these identifiers. The three types differ in the way the overlay is structured [105]:

1. A **centralized peer-to-peer overlay** relies on a central index-server keeping track of the peers and their responsibilities. New peers register with the index-server, and negotiate the set of identifiers they will be responsible for. Queries are issued at the index-server, and the index-server responds with the peers being responsible for the queried identifiers.

A popular example for centralized peer-to-peer overlays is the file sharing platform Napster[5].

2. An **unstructured peer-to-peer overlay** does not require a central index server. Rather, each peer maintains a list of known peers. Queries are broadcasted to the known peers. Subsequently, these peers broadcast the queries to their respective known peers. That way, queries are recursively flooded through the network.

   A popular example for unstructured peer-to-peer overlays is the open source project **Gnutella** [96].

3. A **structured peer-to-peer overlay** implements the **distributed hash table (DHT)** approach. A DHT protocol establishes an overlay topology where each peer is responsible for a range of identifiers. The peers in a DHT maintain routing tables. When a set of identifiers is queried, the routing tables are used to address the peers being responsible for these identifiers. That way, DHTs do not require a central index server, while they still provide efficient routing.

   An example for a popular DHT algorithm is Chord [106], which provides a one-to-one mapping of single identifiers to single peers.

Besides the pure implementations, hybrid peer-to-peer overlays are available, representing mixtures of the topologies above. A very popular example is **JXTA** [43], which provides basically a centralized overlay, but replaces the single index-server with a ring of so-called **super peer**s. The super peers themselves are organized like a simple DHT, where each super peer knows each other super peer.

Motivation for the Focus on Structured Overlays

The use cases studied in this thesis imply the use of structured overlays as the underlying infrastructure. The reason for this is as follows:

As described above, unstructured peer-to-peer overlays use flooding-based query algorithms. This approach generates a large amount of traffic for each peer, while it does not guarantee that all available peers are found [61]. Even though large parts of the network can be flooded with a query, it might happen that there is a responsible peer for a queried identifier that is not reached with the flooded query. As guaranteed access to the queried peers is essential in all of the use case scenarios in this thesis, unstructured peer-to-peer overlays cannot be considered.

[5] `http://www.napster.com`, 24 October 2007

The centralized approach provides guaranteed access to all peers, and it is easy to handle joining and leaving peers in centralized overlays. However, centralized topologies cannot be used in the scenarios in this thesis for two reasons. Firstly, centralized overlays require that somebody operates the central index server, which is a single point of failure. In some scenarios, it is too costly for the participants to run that server at the required level of reliability. Therefore, fully decentralized approaches are preferred.

Secondly, the central index server must process all queries, which results in high traffic load. Again, this makes the server expensive to operate, and these costs can be saved when using fully decentralized overlay topologies.

Providing guaranteed access to responsible peers while being fully decentralized, self-organizing, and resilient to node failure, structured overlays are chosen in the remainder of this thesis.

2.3.4. Summary of Peer-to-Peer-Based Decentralization

The paragraph above introduced peer-to-peer computing as a way to implement decentralization in distributed systems. The term peer-to-peer computing was defined, and different ways of implementing peer-to-peer overlays were presented. In the remainder of this thesis, the focus will be on structured overlays, as they enable full decentralization while providing guaranteed access to responsible peers.

2.4. Summary

This section introduced the background of this work in distributed computing and defined the scope of this thesis in that area. The related topics that are introduced in the first part are chosen with respect to the title of this thesis: ‘‘service layer components for distributed applications’’. Each of the phrases triggers certain associations with related fields, and these related fields were introduced and differentiated from the present work.

In the second part of this section, peer-to-peer computing was introduced in greater detail, as the decentralization in this thesis relies on peer-to-peer overlays as the underlying routing infrastructures. An overview of peer-to-peer computing was given, and it was motivated why structured overlays are chosen as the overlay topologies in the remainder of this thesis.

In the next section, an architecture for peer-to-peer-based decentralized applications is presented. Based on this architecture, a methodology will be derived that helps to identify the gap between application requirements and

state-of-the-art research. This methodology will be applied to different use case scenarios in the following sections.

3. An Architecture for Peer-to-Peer-Based Decentralized Applications

3.1. Requirements Analysis

This section introduces a layered architecture for decentralized peer-to-peer-based applications. The architecture has been derived from the requirements analysis in Section 4. However, the resulting architecture is anticipated here in order to provide a continuous presentation of the use case requirements and the service layer components in the following sections.

The main result of the requirements analysis in Section 4 is: Although the use case scenarios are very heterogeneous, it turns out that most requirements are not application specific, but appear in different scenarios in a similar way.

Therefore, the architecture presented here introduces a component-based service layer. The modular service layer enables the implementation of generic, domain independent components that can be re-used in different applications. The goal is to exploit the repetitive nature of the requirements, and to allow developers to transfer existing solutions to new application domains in an easy way.

The breakdown of peer-to-peer based applications into service layer components enables developers to identify the gap between the application requirements and the properties of the underlying peer-to-peer infrastructure. Studying interdependencies among the service layer components facilitates the evaluation of new peer-to-peer based applications.

3.2. Decomposing Peer-to-Peer Applications

The current situation is as follows: Peer-to-peer algorithms have become a wide research topic in decentralization in the last years [62]. A lot of algorithms and overlay infrastructures have been published. However, there are still only a few application domains benefiting from the potentials offered by peer-to-peer-based decentralization.

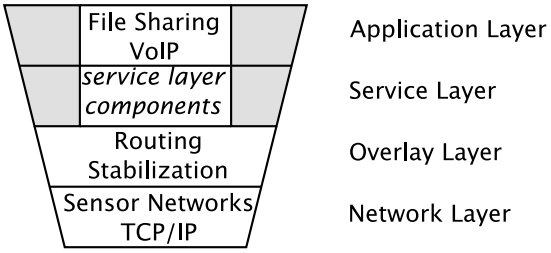


Figure 3.1.: Layers for Peer-to-Peer Based Applications [102]

The reason for this becomes clear when decomposing peer-to-peer applications into different layers, as illustrated in Figure 3.1 [102]. The raw peer-to-peer overlay does not match the requirements of real-world applications, e.g. the raw overlay does not natively fulfill security requirements, or requirements for data consistency[1].

Therefore, additional functionality must be added in order to fulfill the application requirements. However, although the overlay layer has been a research topic for several years, there is no significant related work addressing this additional functionality as independent building blocks that can be composed in order to form decentralized infrastructures meeting the application requirements.

In this thesis, a service layer is introduced, fulfilling application requirements on top of the raw peer-to-peer overlay. The service layer is composed of generic software components that can be put together in order to provide the the functionality required by the applications.

In Figure 3.1, the current situation is illustrated in white. The lack of research on the service layer results in a low number of applications using peer-to-peer [102]. As shaded in gray, the ideal situation should include a larger number of well-defined and integrated service layer components, enabling more applications to be built on top of that.

The reason for the gap between the application requirements and the service layer is not that there are too few algorithms for service layer components available. Rather, the problem is that current peer-to-peer projects tend to provide monolithic, domain specific solutions, and do not distinguish generic service layer components.

Examples for this are Tapestry, which natively includes replication [117], Pond, which guarantees data consistency [86], and the RMF, providing access control [93]. The merge of application-specific functionality and the raw overlay protocol makes it hard to benefit from state-of-the-art peer-to-peer research

[1] More examples of application requirements are presented in Section 4.

in new application domains, such as the application scenarios described in Section 4. An example for that was given in Section 1.

The goal of this thesis is to present an architecture enabling the establishment of a methodology for transferring related work to new fields of application. The methodology helps the developer to identify the gap between state-of-the-art research in peer-to-peer, and the application requirements. This methodology is derived from the analysis of three different use case scenarios, and for each of these use cases the gap is closed, and new service layer components are provided.

3.3. The Architectural Layers

3.3.1. Overview

In this thesis, a service layer is claimed, complementing the raw peer-to-peer layer. The service layer provides generic, domain independent components, providing data consistency, security, resilience, etc. The component-based approach is new in peer-to-peer related research. Current applications either use the raw overlay directly, or build on application-specific, monolithic APIs [28]. Section 1 presented Tapestry as an example of an overlay implementing application-specific functionality.

When defining a service layer between the application and the overlay, there is no obvious line to be drawn between the pure overlay and service layer functionality. Many service layer components are tightly coupled with the underlying overlay algorithm.

Approaches for Defining the Layers

When drawing the line between the overlay layer and the service layer, the goal is to make these layers as loosely coupled as possible. The following approaches can be taken to achieve this goal:

1. Put as much functionality as possible into the overlay layer. This is basically what current research on peer-to-peer does. The results are large monolithic infrastructures, including load balancing, access control, accounting, etc. However, interdependencies of the different functionalities can be handled efficiently in monolithic applications.

2. Put as little functionality as possible into the overlay layer. This approach is taken in this thesis. The advantage of keeping the overlay layer small is that many of the building blocks of peer-to-peer-based applications

become located in the service layer, i.e. many of the interrelations between these building blocks become interrelations within the service layer, and do not affect the interface between the service layer and the overlay layer. That way, the relation between service layer components and the peer-to-peer layer becomes less tightly coupled.

Both approaches have advantages and disadvantages. While the first approach yields efficient implementations that are tailored to specific application scenarios, the second approach helps to decrease development time, as it facilitates transferring functional components to new fields of application. The goal of this thesis is to foster the adoption of peer-to-peer-based decentralization in industry. Based on the second approach, a methodology is developed that enables developers in industry to benefit from state-of-the-art research in peer-to-peer-based decentralization.

Definition of the Layers in This Thesis

Figure 3.1 shows the layered architecture as proposed in this thesis. The following list introduces these layers in bottom-up order:

- The **network layer** is commodity in most cases, except for some specific applications, such as applications based on sensor networks. In the application scenarios described in Section 4, the network layer is either TCP/IP or a VPN on top of TCP/IP.
- The **overlay layer** covers the decentralized peer-to-peer infrastructure. As motivated in Section 2, the focus of this thesis is on DHTs as peer-to-peer overlays. Many well-researched algorithms are available on the overlay layer. A detailed overview of the most common DHT algorithms is given in Appendix A.
- The mere routing functionality provided by the overlay is enriched on the **service layer**. The service layer is composed of generic, domain independent service layer components, adding features like data consistency, confidential communication, or support for range queries.
- On top of the service layer, the **application layer** implements the actual application, using the customized interface provided by the service layer.

3.3.2. Overlay Layer

The approach in this thesis is to put as little functionality as possible into the overlay layer. The raw peer-to-peer overlay provides only two functionalities:

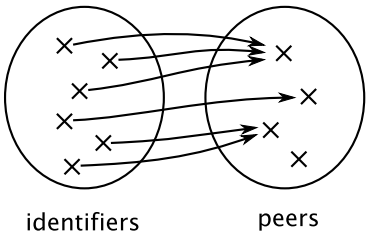


Figure 3.2.: Lookup operation in Chord: 1 identifier is mapped to 1 peer.

routing and the stabilization.

- **Routing**. Each peer is responsible for a set of identifiers. The routing implements a mapping between the identifiers and the responsible peers. One can use the routing algorithm to query a set of identifiers. The query is routed to a set of peers being responsible for these identifiers.
- **Stabilization**. Unlike computer clusters, peer-to-peer systems are loosely coupled distributed systems. Peers may continuously appear and disappear, which is called **churn** in peer-to-peer terminology. The purpose of stabilization is to keep the routing functionality intact in spite of churn.

The following terms have been used in the descriptions above:

- **Identifier**. In most peer-to-peer applications, the identifier is a keyword or hash value associated with data to be stored in the peer-to-peer overlay. However, the definition above leaves the use of the identifier open. For example, the identifier could also be associated with a computational task in a distributed computing application.
- **Set of identifiers**. Peer-to-peer overlays can be used to query a set of identifiers. The definition above leaves it open how many elements this set may have. Many peer-to-peer overlays allow only queries for single identifiers, as in Chord [106], which is illustrated in Figure 3.2. Others support multiple identifiers, but require the number of identifiers to be fixed, e.g. CAN [84]. There are also peer-to-peer overlays natively supporting queries for arbitrary sets of identifiers, like Gnutella [96].
- **Set of peers**. As with the set of identifiers, the definition above leaves it open how many peers are responsible for a certain set of identifiers. Many peer-to-peer overlays provide a one-to-one mapping between single

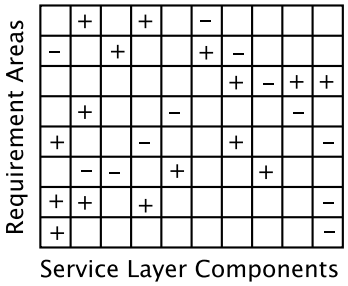


Figure 3.3.: Example Layout for a Map of Service Layer Components

identifiers and single peers, as in Figure 3.2. But there are other implementations where many peers are responsible for a certain identifier, e.g. in file sharing, where many peers might serve the same MP3 file.

To summarize the definition above: The overlay layer provides is a distributed routing infrastructure that can be used to map a set of identifiers to a set of peers being responsible for these identifiers. The overlay provides a stabilization mechanism that allows the routing infrastructure to adapt to changes when peers join or leave the network.

3.3.3. Service Layer

In the architecture description above, a service layer was proposed as an intermediate layer between the peer-to-peer overlay and the application. The service layer is to be composed of generic, domain independent service layer components, each fulfilling some requirements of the application.

However, Section 5 will show that service layer components cannot be put together arbitrarily. Rather, they interact with each other, and some combinations may corrupt the components' functionality.

In order to foster the component-based approach, a map and classification of service layer components is provided as part of this thesis. In Section 4, use cases from different application domains are analyzed. In the following sections, existing algorithms as well as new algorithms will be investigated, and transformed into building blocks that can be used as service layer components to build applications meeting the requirements found in the use case analysis.

The experiences from the use case implementations will be collected in a map of service layer components. Figure 3.3 shows an example of how a map of service layer components looks like. A plus symbol indicates that a component

supports a specific requirement area, while a minus symbol indicates that a component has a negative effect on that requirement area.

The presentation of the map will show the impact of the service layer components on each other, as well as the impact on the requirements of different application domains.

Comparison to Related Work

The definition above leaves some tasks to the service layer that use to be part of the peer-to-peer overlay in other peer-to-peer related publications. The motivation for this is to decouple the overlay layer and the service layer, and to handle all interdependencies within the service layer. In this paragraph, some examples are given for functionality that can be found in the overlay in related work, but that is not found in the overlay according to the definition in this thesis.

Most publications on peer-to-peer overlays include instructions on how to assign the identifiers associated with each peer, e.g. in Chord peer identifiers are chosen by hashing the node's IP address [106]. Leaving this to the service layer makes it possible to choose the right strategy independently of the underlying peer-to-peer overlay. For example, in order to facilitate load balancing, one could choose the identifiers dynamically to relieve hot spots [88, 65].

There are other examples of functionality that is often associated with peer-to-peer overlays, but that does not belong to the peer to peer layer according to the definition above. One of the most important examples is replication. Leaving the replication strategy to the service layer makes it possible to consider the implications on other service layer components independently of the underlying peer-to-peer overlay.

3.4. Summary & Outlook

In this section, a layered architecture for peer-to-peer-based decentralized applications was proposed as an alternative to monolithic applications. A service layer was introduced, fulfilling application requirements on top of a lightweight overlay layer. The service layer is composed of generic, domain independent service layer components that can be used as building blocks to form peer-to-peer-based infrastructures providing customized, application-specific APIs.

The architecture presented in this section is derived from the results of the analysis of three industrial applications that will be introduced in the next section. The main outcome of the analysis is that many requirements appear

in different scenarios in a similar way, despite the diversity of the application domains.

The next section presents the requirement analysis. Section 5 reviews related work, and identifies components for the requirements found. For most of the requirements, service layer components can be formed using ideas from related work. However, some requirements in the use case scenarios are not yet solved, and new service layer components need to be developed. These are presented in Section 6. The overall results are evaluated in Section 7.

4. Application Scenarios

In this section, three industrial application scenarios for peer-to-peer-based decentralization are introduced, and the requirements are analyzed. As anticipated in Section 3, it will turn out that many requirements appear in several scenarios in a similar way, in spite of the diversity of the application domains. This observation leads to the component-based architecture of the service layer that has been introduced in Section 3.

4.1. Requirements for Scenario Selection

Defining the basis for new software architectures is always a trade-off. If the basis is too broad and general there is a risk of creating architectures with limited practical relevance. On the other hand, when deriving an architecture from specific use cases, there is a risk that the resulting architecture is only valid for a limited set of applications.

When choosing the application scenarios for this thesis, the focus was on practical relevance. Therefore, the architecture has been derived from three specific use cases. In order to keep the results as general as possible, the use cases are taken from different application areas. The following criteria were taken into account when choosing the scenarios:

Diversity. The use cases are chosen from a wide range of application areas, covering different domains. That way, as many aspects as possible are taken into account.

Multiple Impact. The experience gathered from the use case analysis must be generic enough to be transferable to other applications and other application domains. That way, the overall contribution of the results is maximized.

Business Relevance. At the time of the investigation, all use cases were part of the Siemens business. That way, the identified requirements could be discussed and verified with professionals from the chosen application domains, and the results can be applied in future Siemens projects.

The next subsection provides a quick overview of the three application scenarios building the basis for this thesis, and motivates the choice of the use cases with respect to the criteria above. The following subsections present the scenarios in detail.

4.2. Scenario Walkthrough

Before the three application scenarios are introduced in detail in the next sections, a quick overview is provided, showing the diversity, multiple impact, and business relevance of the use cases:

The business collaboration platform presented as the first use case is found in the application domain of business integration. It incorporates a peer-to-peer overlay with a relatively small number of users. The focus is primarily on security, such as confidential communication. The scenario has been developed as part of the **ATHENA**[1] project, which is an Integrated Project funded by the European Commission, aiming at business interoperability. Siemens participates as one of the industrial partners in ATHENA.

The searchable user directory scenario presented as the second use case describes an Internet-scale application, providing support for a very large number of users. The primary focus is on scalability. The scenario has been developed as part of Siemens' **Peer Things**[2] project.

The distributed power generation scenario presented as the third use case has been developed in cooperation with Siemens Power Transmission and Distribution (PTD) Group. The focus is on providing a reliable control infrastructure to operate the power generators.

All of the scenarios above are related to Siemens projects. They have different focuses, while the main characteristics of the use cases are generic enough to be relevant for other peer-to-peer based applications, as well. Therefore, the choice of these use cases satisfies the requirements motivated above.

[1] http://www.athena-ip.org, 2 April 2007
[2] Peer Things was presented at the CeBIT trade show in 2006, see http://www.p2p-blog.com/?itemid=76, 1 December 2007

4.3. Business Collaboration

The first use case describes a business collaboration scenario in the automotive industry. The integration of heterogeneous enterprises plays an increasingly important role in the globalized market[3]. Enabling the integration of business processes is one of the prerequisites when different organizations transform themselves into **networked organization**s.

The application scenario described here was originally developed in the context of the ATHENA project. A key outcome of the analysis of business process interoperability was that there are two different kinds of collaboration paradigms [103]:

1. The **process-driven collaboration paradigm** is characterized by long-term collaborations with a rarely changing set of partners. The business processes are usually orchestrated by a central management instance, which results in hierarchically organized IT infrastructures as the basis for these scenarios.

2. In the **event-driven collaboration paradigm**, notable things are disseminated immediately to all interested parties. The interested parties evaluate the event, and optionally take action [64]. Event-driven processes are asynchronous, dynamic, and document-centric. The business partners join and leave the collaboration very dynamically. The lack of a central process lead on the business level fosters the need for a decentralized collaboration platform to be deployed on the IT level.

While there have been significant research efforts on process-driven collaboration scenarios [113], IT support for event-driven collaboration paradigm is a new research topic. The next section presents the application scenario that was developed as part of the ATHENA project. The scenario contains both an example of process-driven collaboration as well as a typical example of event-driven collaborations.

4.3.1. Application Scenario

This subsection presents a scenario showing both a typical example of the process-driven collaboration paradigm and an example being representative for the event-driven collaboration paradigm. The scenario is is taken from a car manufacturers strategic sourcing process. Strategic sourcing is an institutional procurement process that continuously improves and re-evaluates

[3] See "What is ATHENA" on `http://www.athena-ip.org`, 2 April 2007

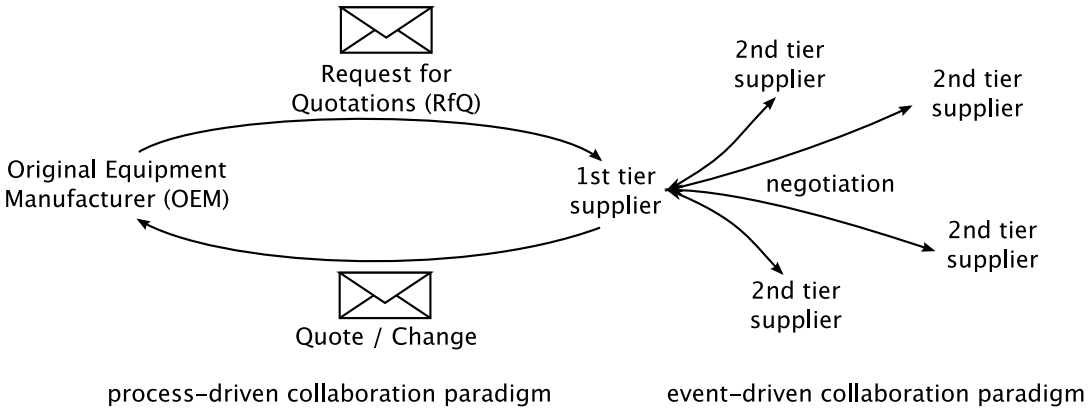


Figure 4.1.: Paradigms in Collaborative Product Design

the purchasing activities of a company. It is one component of supply chain management[4].

As illustrated in Figure 4.1, the car manufacturer (**original equipment manufacturer (OEM)**) issues a **request for quotations (RfQ)**, and submits the RfQ to one of its first tier suppliers. In turn, the 1st tier supplier negotiates with the 2nd tier suppliers and creates a response, typically containing some change requests to the original specification contained in the RfQ. Then, the 1st tier supplier sends the change request to the OEM.

The OEM integrates these changes into the overall specification and issues a revised RfQ document. This cycle continues until the OEM and the 1st tier supplier agree on a final specification. This process is part of the **collaborative product design (CPD)** or **collaborative product development**. The final specification of the product (the car) is defined jointly by the collaborating business partners.

Relations between the OEM and 1st tier suppliers are of a long-term nature. Business partners join or leave the collaboration infrequently, which makes the collaboration on the OEM side a typical example for the process-driven collaboration paradigm. On the supplier side, a typical example for event-driven collaboration can be found. The process is centered around the RfQ document, and actions are triggered by changes to that document [103]. Many 2nd and higher tier suppliers are **small and medium enterprises (SMEs)**, joining and leaving the supplier relationships very dynamically.

The decentralized structure and the dynamics of event-driven business collaboration makes decentralized peer-to-peer computing a promising approach

[4] http://en.wikipedia.org/wiki/Strategic_sourcing, 13 November 2007

for implementing event-driven collaboration platforms [103]. In the following, the requirements for the CPD scenario are presented.

4.3.2. Application Requirements

The following list presents the requirements that need to be satisfied when developing a decentralized peer-to-peer-based business collaboration platform for CPD. The requirements analysis is based on the results of the ATHENA project. Eight requirement areas have been identified [102]:

Messaging. As explained above, the collaboration platform must not only serve as a data store for business documents. It also needs to notify the business partners if relevant documents are added, updated or removed. Doing so requires some messaging functionality.

Reliability. Business partners are SMEs, joining and leaving the supplier relationships very dynamically. The constant change of participating collaborators must not affect the interaction of other business partners. The collaboration platform must be resilient to churn.

Self-organization. In order to keep maintenance costs as low as possible, the system should provide plug-and-play functionality, which means that new business partners should be able to join the network without the need for the other partners to re-configure their software manually.

Limited network load. The peers in the peer-to-peer infrastructure are all operated by the participating business partners. The expected number of partners using the collaboration platform is relatively low, as compared to other peer-to-peer-based applications, e.g. the distributed user directory. However, the documents being exchanged might be relatively large. Each peer acts as a router for other peers. The network traffic should be equally distributed among all peers. The platform must avoid that a single peer is flooded with all the traffic.

Limited data load. As the business documents might be comparably large, care must be taken to limit the data load to be stored on each single peer. In the ideal case, each peer should store roughly the same amount of data.

Data consistency. Due to the decentralized nature of peer-to-peer systems, there is no central instance defining which version of a document is the current one. Therefore, it is required that the collaboration infrastructure provides means for maintaining a consistent view and versioning of all resources.

Security. The ITSEC Standard defines three aspects of IT security [24]:

- confidentiality -- prevention of the unauthorized disclosure of information.
- integrity -- prevention of the unauthorized modification of information.
- availability -- prevention of the unauthorized withholding of information or resources.

Business partners may be both collaborators and competitors at the same time. Therefore, confidentiality must be guaranteed. Additionally, data integrity must be guaranteed, i.e. data stored in the network must be encrypted and resistant to malicious modifications or removal. Data availability is also an important factor. This aspect is partially related with the reliability requirement above.

Rich queries. There must be a way for business partners to specify documents they are interested in. The underlying collaboration platform must offer some rich query language for formulating complex queries.

4.3.3. Summary

In this subsection, a CPD scenario from the automotive industry was presented, and the need for a peer-to-peer based business collaboration platform on the supplier side was motivated. The CPD scenario involves a relatively small number of participants, and the collaboration platform is built on top of a closed VPN. The decentralized approach provides a high level of reliability in spite of the churn of the business partners. The focus was on the confidentiality and messaging functionality. In contrast, the searchable user directory scenario below describes an application that requires Internet-scale deployment with a very large number of users.

4.4. User Directory

While the business collaboration scenario presented above targets business-to-business transactions, the application scenario introduced here is a service for end users. The searchable user directory is an example of an Internet-scale application, where the focus is on high scalability, with an estimated number of 500 000 users and more. The high level of scalability can be achieved applying a decentralized peer-to-peer overlay as the basis for the application.

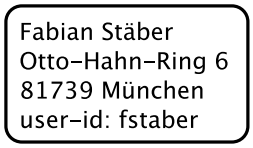


Figure 4.2.: Example of a Phone Book Entry

4.4.1. Application Scenario

Besides file sharing, IP telephony has become the most widely used peer-to-peer-based application. Phones with built-in peer-to-peer stacks are used to enable IP telephony in closed networks on large company sites as well as in the Internet.

As part of the Peer Things project [92], the Siemens Communications Group[5] developed a decentralized communication platform supporting video communication, voice communication, instant messaging, etc. While parts of Peer Things require centralized services[6], the communication platform implements many features in a decentralized way, using an underlying DHT.

In Peer Things, each user is associated with a unique identifier, e.g. a phone number or a nickname. In order to contact a user in Peer Things that user's identifier must be known. Peer Things uses the DHT to map the identifier to the current IP address and port number of that user.

However, users do not always know the unique identifier of the person to be contacted. Therefore it must be possible to look up the identifier in a user directory. This directory is required to support range queries, like queries for all users with a certain last name.

The application scenario presented here describes a searchable user directory that can be integrated into the Peer Things platform. The user directory will be implemented in a decentralized way in order to provide a high level of scalability and resilience.

An example for a directory entry is shown in Figure 4.2. A query for the last name "Stäber" should retrieve all entries with that last name. The unique user identifier stored with each entry can then be used to access Peer Things' communication facilities.

Implementing a decentralized user directory with range query support is

[5] The Siemens Communications Group was transferred to Nokia Siemens Networks recently, so the Peer Things Web site is currently not available. However, information can be found on the Peer Things presentation at the CeBIT trade show in 2006, e.g. http://www.p2p-blog.com/?itemid=76.

[6] For example, when deployed in the Internet, VoIP telephony must allow lawful interception [68].

yet an unsolved challenge [101]. The service layer components presented in Section 5 and 6 provide the building blocks that can be used to meet this challenge.

In the following section, the requirements for the user directory are derived from the results of the Peer Things project. These requirements will be the basis for the evaluation of the service layer components in Section 7.

4.4.2. Application Requirements

As part of the Peer Things prototype specification [92], requirements for the user directory were defined:

Scalability. In the first phase of the Peer Things project, up to 500 000 users were to be supported [15]. Later releases targeted a number of more than 1 000 000.

Self-organization. With the large number of users, it becomes very costly to manage the entries manually. Therefore, self-organization is required, such that users can join or leave the directory without the need for manual interaction.

Reliability. One of the core requirements in the Peer Things project was that entries must not become inaccessible incidentally.

Limited data load. The hardware used as peers are DSL routers based on embedded boards, which implies limited storage capabilities. As an estimate, a single peer cannot store more than 300 entries.

Limited network load. As peers are connected via regular phone lines, the network load must be well balanced in order to avoid high peaks of network loads on single peers.

Rich queries. Just like a printed phone book, the user directory must allow range queries, i.e. it must be possible to list all entries with a certain last name. While range queries are a very simple requirement as compared to other data mining concepts, e.g. Ontologies, it is still challenging to support efficient range queries in decentralized systems. The reason for this is that DHTs natively only support routing functionality for exact searches.

Sparse population. The requirement for range queries above is complemented by another requirement: Each phone client is associated with exactly one entry in the user directory. Therefore, the number of entries equals the

number of peers, which results in many peers not being responsible for any data at all. This raises the danger that resolving queries takes too long, as many empty peers might be connected needlessly. The user directory must be able to handle the empty peers efficiently, and to resolve queries efficiently in spite of the sparse population.

Security. In the requirements analysis of the CPD scenario above, the three aspects of IT security defined in the ITSEC standard were used [24]: confidentiality, integrity, and availability. In the user directory scenario, confidentiality does not play a role because data in a phone book is public. However, data integrity is an important issue, as it must be guaranteed that entries are not altered maliciously, because this would allow an attacker to redirect arbitrary phone calls. The aspect of availability is covered by the reliability requirement above.

The requirements above are only the requirements having direct impact on the distributed user directory. The Peer Things project provides solutions for a lot of additional requirements. For example, a NAT[7] traversal mechanism [34, 46] was implemented to deal with users accessing the Internet through NAT routers. The distinction of active and passive peers was introduced to deal with users with very short mean online times. Lawful interception was enabled to comply with legal requirements.

However, this thesis focuses on the yet unsolved problem of implementing a decentralized user directory with range query support.

4.4.3. Summary

Above, a decentralized, searchable user directory was introduced as the second application scenario for this thesis. Unlike the business communication scenario before, the user directory has its focus on Internet-wide scalability, involving a very large number of users.

The third and last application scenario introduced below will have its focus on messaging functionality in a reliable environment.

4.5. Distributed Power Generation

When choosing the use case scenarios for this thesis, attention was payed to covering a wide range of application domains. The business collaboration scenario is designed for a comparably small number of participants in a closed

[7] NAT: Network Address Translation

VPN. The user directory is an application to be deployed in the Internet, with the focus on scalability. The third scenario is taken from the electric power industry, and describes a distributed control infrastructure [100]. The focus of the power generation scenario is on reliability.

By today, electric power is mostly generated by a few large power plants, and distributed from these plants to the consumers. This situation is changing dramatically. The growing use of alternative energies, like solar energy, wind power, and biomass energy results in an increasing number of very small power generators. The organization of power generation is shifting from a central structure towards distributed power generation [112].

The new distributed power generation scenario raises the need for a reliable infrastructure to control the power generators. For example, this control infrastructure is necessary to inform biomass energy generators to reduce their production whenever sufficient wind power is available, and to raise the amount of biomass energy when the generated wind power decreases. The control infrastructure must be highly scalable, as every single household might potentially have devices for generating power or regulating power consumption in the future.

4.5.1. Application Scenario

The shift towards decentralization in power generation raises the requirement for a self-organizing Internet-scale control infrastructure. The infrastructure must be resilient to node failure, as unreachible generators might produce power that is not needed, resulting in a waste of resources and in increasing costs for the utility. This subsection shows how the peer-to-peer paradigm can be applied to implement such an infrastructure. It presents the application scenario for a peer-to-peer-based reliable control infrastructure for distributed power generation.

There are two parties involved: The **utility** runs several hosts that are connected to the Internet, and manages the control infrastructure. The **power generator**s are connected to the control infrastructure. The utility uses the control infrastructure to send control commands to the generators, and to query status information from the power generators.

In addition to providing control of the power generators, the infrastructure should be scalable enough to enable every single household to receive control commands from the power generators. Firstly, with the growing use of alternative energy, it might be possible that a large share of the households provides some devices for generating power in the future. Secondly, it might be desirable to regulate power consumption to achieve better balancing on peak times. For

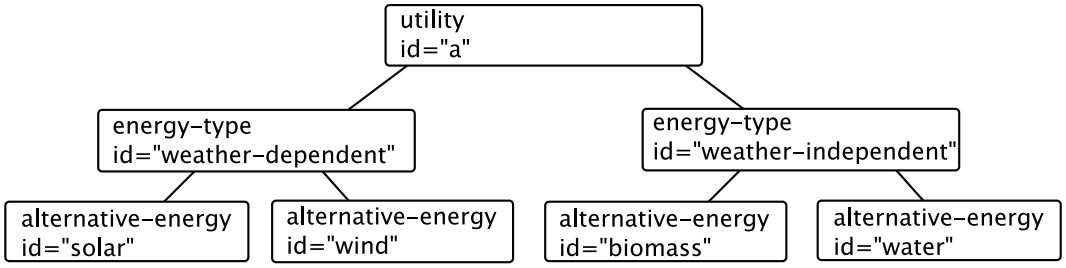


Figure 4.3.: Energy-Type-Based View of the Power Generators

example, the infrastructure might be used to inform heatings to reduce their power consumptions on peak time.

When the infrastructure is set up, the power company starts with only a few peers. With a growing number of power generators being connected to the infrastructure, the power company sets up new peers, and the load is balanced among these peers.

From the user's point of view, the control infrastructure provides tree-shaped data structures, being resilient to node failure. The power generating devices are logically located at the leaf nodes of the tree structures, and the power company uses the trees for addressing the power generators. The tree structures are provided independently of the underlying hardware. As each tree node is physically replicated on several peers, even if underlying hosts fail, are added, or removed, the tree structures remain available.

The control infrastructure is capable of holding several of these tree structures in parallel. That way, the power company can choose the most convenient view depending on the use case. Examples for tree structures are illustrated in Figure 4.3 and Figure 4.4.

Figure 4.3 shows a view that is organized by the energy type. Power generators can be divided into two categories: Generators depending on the weather conditions like wind power generators or solar power generators, and generators that can be regulated independently of the weather condition, like biomass generators or water power generators.

For example, if the wind calms down, the utility might decide to request the biomass generators to increase their output. Using the tree structure in Figure 4.3, a broadcast to all biomass generators can be expressed as follows:

```
/utility[id='a']/energy-type[id='weather-dep.']/alt.-energy[id='wind']/...
```

As another example, Figure 4.4 shows a location-based view of the power generators. If the power company wants to know the current capacity of all

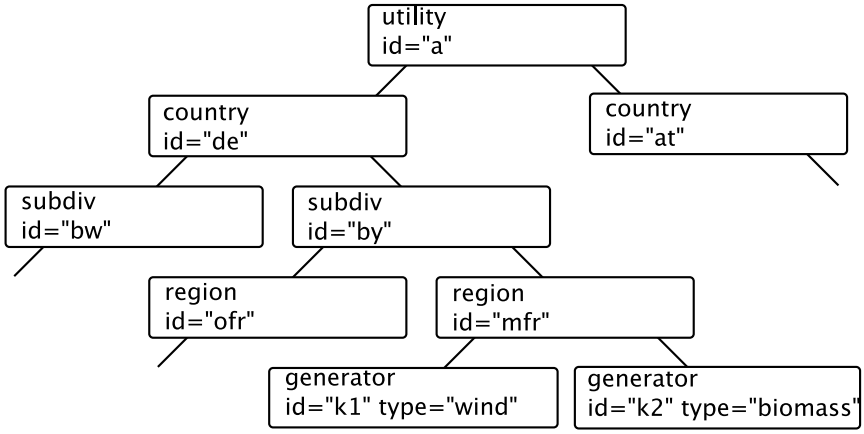


Figure 4.4.: Location-Based View of the Power Generators

wind power generators in a specific region, then it is likely to use a location-based view to broadcast a query to that region. A query to all wind power generators in the subdivision `'by'` could be addressed like

```
/utility[id='a']/country[id='de']/subdiv[id='by']/*/generator[type='wind']/...
```

As a response, the routing infrastructure should return the aggregated data of all wind power generators in the subdivision of `'by'`.

4.5.2. Application Requirements

The following presentation shows the utility's requirements for the control infrastructure:

Reliability. The tree-shaped routing infrastructure must be resilient to hardware failure. Increasing the level of reliability as compared to classical client-server based systems is one of the main motivations for building a peer-to-peer-based control infrastructure.

Self-organization. The logical tree structure should be independent of the underlying physical infrastructure. Utilities must be able to start the deployment with a small number of peers, and increase the number of peers later, when more power generators get connected to the infrastructure. It must be possible to add and remove hardware without the need for adapting the logical tree structure.

Limited network load. Even if the control infrastructure keeps growing, it must be ensured that no single peer is flooded with overly high network load.

Scalability. It is likely that a large share of the households will provide some power generating devices in the future, or that they will at least provide some devices where the power consumption can be controlled to reduce consumption on peak load. The control infrastructure must thus be able to scale to a large number of clients.

Multicasting and aggregation. The control infrastructure must be able to support multicasting when sending control commands from the power company to the power generators, and it must provide an aggregation tree for receiving periodic reports from the power generators.

Messaging. The propagation of events from and to the power generators implies the requirement for messaging capabilities on the peers. Peers must not only store data, as in regular overlays, but also serve as routers for messages.

Security. A security concept must guarantee that only the utility is able to alter the routing infrastructure. The power generators themselves must be clients that cannot modify the tree structure. That way, confidentiality and integrity can be provided. The aspect of availability is covered by the reliability requirement above.

Another requirement often mentioned in related application scenarios is real-time support, providing guarantees for duration constraints for command delivery. However, implementing real-time support is not possible when the application is deployed on top of the Internet, as the Internet does only provide a best-effort service. Implementing soft real-time is still possible on the application layer, sending time-stamps with each command telling the receiver when the command should be executed. However, this is not addressed in this thesis. The focus in this work is on building a reliable messaging infrastructure with multicasting and aggregation support.

4.5.3. Summary

Above, a peer-to-peer based control infrastructure for distributed power generation was introduced as the third application scenario for this thesis. The focus was on reliability and scalability. Unlike the user directory above, the control infrastructure is to be deployed in a closed environment being set up and controlled by the utility.

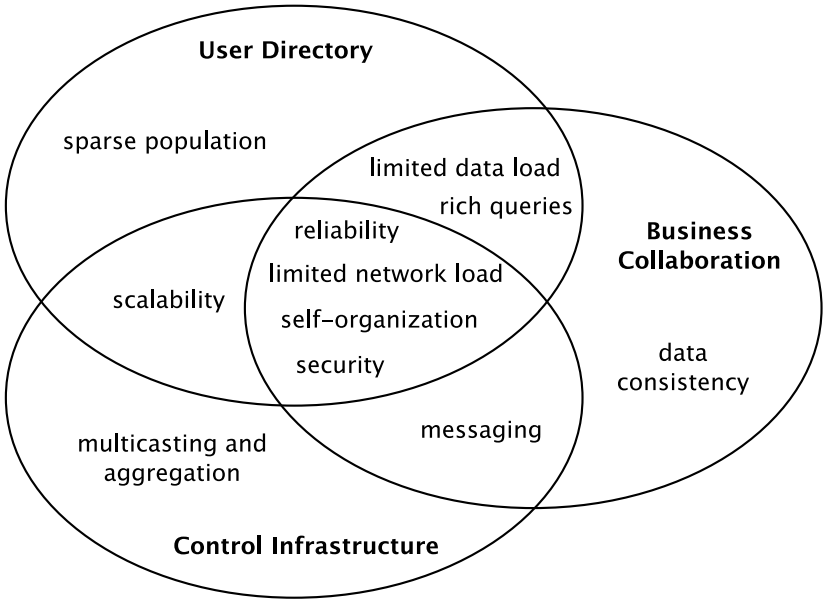


Figure 4.5.: Requirements

4.6. Map of Requirements

In the presentation above, three application scenarios were introduced, and the requirements were presented. The scenarios are chosen from different application domains in order to cover a broad range of requirements. All of the scenarios are related to Siemens projects. That way, the requirements analysis could be backed with the experiences from the corresponding projects.

Figure 4.5 shows the requirements identified above, grouped by the scenarios they appear in. It becomes clear that most requirements have multiple impact, i.e. they are found in more than one scenario. Only a few requirements are specific for a single application only.

Figure 4.5 has to be interpreted carefully. If a requirement area is not included in a specific scenario, the conclusion can be drawn that the implications associated with that requirement are not relevant for that use case. For example, data consistency is considered as not relevant for the decentralized user directory, as concurrent modifications of entries is not required.

On the other hand, if a requirement is found in more than one scenario, no general conclusions are possible. It is possible that the same requirement area implies very different means depending on the use case. For example, security in the business collaboration platform focuses on confidential communication

among the peers, while security in the context of the control infrastructure means preventing malicious routes.

The overall contribution of Figure 4.5 is therefore to provide an overview of which requirements are application specific, and which are relevant for more than one application. The requirements being relevant for more than one application need detailed investigation with respect to the specific use cases. The remainder of this thesis will show that many of the requirements with multiple impact allow the definition of generic software components that can be re-used in different application scenarios.

4.7. Summary

In this section, three different application scenarios were presented, and a map of use case requirements was created. All of the scenarios described above contain yet unsolved challenges. Providing peer-to-peer-based, decentralized solutions for each of these scenarios is part of the contribution of this thesis.

In the next section, state-of-the-art in decentralized peer-to-peer computing is reviewed, and generic, domain independent service layer components are identified. However, for each of the use case scenarios a gap remains between state-of-the-art and the application requirements. Section 6 closes that gap presenting new components that are necessary to implement the applications.

5. Definition and Classification of Service Layer Components Implementing State-of-the-Art Algorithms

5.1. Structure of the Presentations

Reviewing related work on peer-to-peer based applications, it turns out that many of the requirements found in the scenarios in Section 4 do also apply in a similar way to related applications. However, current research does not apply the component-based view of peer-to-peer-based decentralization as proposed in this thesis. Rather, the solutions presented in related work are often interweaved into monolithic applications, and cannot be used instantly as building blocks being available for other application scenarios.

In this section, current research in decentralized peer-to-peer computing is reviewed, and functionality is generalized as reusable service-layer components.

The presentation of the components is organized according to the components' functionality. It turns out that different components providing similar functionality do mostly have similar interrelations with other service layer components.

The structure of the presentations is as follows:

Technology Description gives an overview of state-of-the-art solutions.

Application Example shows how the component can be applied to the application scenarios in Section 4.

Impact Analysis summarizes the impact of the components on the requirement areas identified in Section 4.

Regarding the application scenarios described in Section 4, it turns out that not all of the requirements can be solved using ideas found in related work. Therefore, the components described here will be complemented with new components in Section 6.

5.2. Component Definitions

5.2.1. Component Walkthrough

Before introducing the components in detail, the following list provides an overview of the functionality that is to be discussed:

Replication is a very basic functionality, providing data redundancy. Applying a replication component increases reliability, and enables traffic load balancing.

Consensus Protocols are often used in conjunction with replication in order to provide data consistency.

Public Key Infrastructures provide the basis for implementing security concepts.

Caching aims at better performance and reduced network load.

Subscriptions are used to implement messaging functionality.

Fuzzy Hashing subsumes a number of algorithms aiming at data load balancing.

Redundant Paths protect against certain kinds of attacks on the overlay's routing protocol, and provide network load balancing.

Search Index refers to a variety of concepts aiming at rich query functionality.

5.2.2. Replication

According to the architecture definition in Section 3, the raw overlay layer merely provides routing and stabilization functionality without any replication or resilient data storage. Replication is a service layer component used to store backup copies of each data resource. The group of peers that keep copies of a resource is called **replication group** for this resource. Replication serves two purposes:

1. *Resilience.* In the overlay, each peer is responsible for a set of identifiers. When a peer becomes unreachable, the overlay's stabilization protocol will reconfigure the routing tables, and shift that peer's responsibilities to another peer.

   In a peer-to-peer-based data repository, each data resource is associated with a set of identifiers and stored on the peers being responsible for

these identifiers. The replication protocol stores backup copies of the data on the peers that would take over the associated identifiers in case of node failure.

That way, data remains available in spite of churn. If a peer fails, the replication protocol reorganizes the replication group order to provide sustainable reliability.

2. *Traffic Load Balancing.* Without replication, the peer being responsible for a resource can become a bottleneck if that resource is requested frequently. If the replication group is chosen in a way that the peers in the replication group are likely to be on the routing path to the replicated resource, queries can not only be responded by the responsible peer, but by any peer in the replication group. That way, the overall network traffic is reduced, and the traffic is balanced among the peers in the replication group.

Replication is a very common functionality being natively included in most peer-to-peer implementations. However, different replication strategies have different implications on the application layer. Therefore, it is worthwhile to separate replication from the raw peer-to-peer overlay and implement it as a generic service layer component. That way, the application developer can choose the replication strategy that fits best with the given application requirements. An example of the different implications of replication strategies is shown in the evaluation of the distributed power generation scenario in Section 7.

Technology Description

Replication is an essential component, and in current DHT implementations replication is often tightly coupled with the underlying peer-to-peer layer, as exemplified in Section 1. In the layered architecture presented in Section 3, the interface provided by the peer-to-peer layer was defined as a mapping of a set of identifiers to a set of peers.

If there is more than one responsible peer per identifier, replication can be implemented easily, because the data can simply be stored on each of the peers being responsible. An example for this is the RMF [93]. It implements a Chord-like DHT on the peer-to-peer layer, but retrieves 3 neighboring peers for each queried identifier. Therefore, in the RMF, replication can be implemented as a lightweight service layer component storing resources on these neighboring peers. Other examples are Pastry [91] and Tapestry [117], which retrieve the peers on the routing path for each queried keyword. Here, the data can be

replicated on the peers on the routing path, which additionally accelerates consecutive lookups using the same path.

If there is only one responsible peer for each set of identifiers, or if more sophisticated replication strategies are required, then more complex service layer support is necessary. There are a lot of sophisticated strategies for implementing the service layer components. As an example, dynamic replication strategies can be used which adapt to the popularity of given resources. DHash [27] and Beehive [83] are well-known implementations of this. A good survey and evaluation of different strategies can be found in [59].

Application Example

The control infrastructure for distributed power generation described in Section 4 relies heavily on replication to provide the required level of reliability. Here, the resource keeping peers are responsible for routing control data on a path being specified in the resource. If one of the peers fails, another peer in the replication group takes over the routing. In the evaluation in Section 7, it is shown that different replication strategies have different implications on the level of reliability achieved.

One of the main results in Section 7 is that using replication, a significantly higher level of reliability can be achieved as compared to regular master/slave architectures. The level of reliability will be evaluated.

Impact Analysis

The figure below summarizes the effects of replication on the different requirement areas identified in Section 4. Obviously, the level of *reliability* is increased when using replication, as data is stored redundantly in the DHT. The evaluation in Section 7 will derive a formula showing the level of reliability that can be reached with replication, as compared to classical client-server-based systems.

Replication has a positive effect on *limitations for network load* if requests are balanced among all peers in a replication group. In that case, there is no single peer that must serve all requests for a certain identifier. This effect is similar to the use of caching, which is described below.

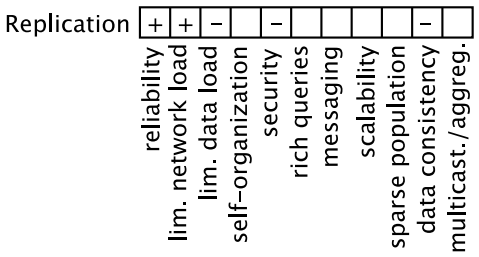
On the other hand, *limitations for data load* might conflict with the use of replication, as data is stored redundantly. The amount of additional data to be stored depends on the size of the replication group.

Most replication strategies work in a completely decentralized way [59], i.e. the use of replication does not affect the *self-organization* properties of the overlay. The impact on *security* depends very much on the use case, but the

developer must keep in mind that there might be negative security implications if there are "floating responsibilities" in the overlay.

In the experiences made with the use case scenarios in this thesis, replication had no effect on the support for *rich queries* and *messaging*. Also, the *scalability* is not affected by replication, as the size of the replication group is fixed and independent of the size of the overall overlay. Replication might have a positive effect on *sparsely populated* overlays, as the overall amount of data is increased. However, the size of a replication group is fixed, and in large-scale deployments this effect becomes negligible. Therefore, no such effect was experienced in the user directory application, which was evaluated simulating more than 500 000 peers.

The negative effect on *data consistency* is explained in the next paragraph, when consensus protocols are introduced. *Multicasting and aggregation* was not affected by replication in the context of this thesis.



5.2.3. Consensus Protocols

Above, replication was introduced as a way to provide resilience and traffic load balancing. However, replicating documents raises the risk of concurrent modifications. Consensus protocols can be used to avoid inconsistencies resulting from concurrent modifications.

Technology Description

In case of churn, one cannot guarantee that all peers have the same view of which peer is responsible for a certain resource. For example, if the peer being responsible for a resource becomes temporarily unreachable, another peer in the replication group takes over the responsibility. If the original peer becomes available again, there are two peers considering themselves as responsible for this resource. This ambiguous situation is fixed after a while by the DHT's stabilization algorithm.

If concurrent modifications are allowed, these modifications could take place on two different peers within the replication group. That way, the system might end up in an inconsistent state, with two different versions of the same resource.

In most applications this can be avoided, as concurrent modifications do not need to be allowed. It is often sufficient to allow only the originator of a resource to modify the resource. The other peers only need read-only access. Moreover, instead of modifying an existing resource, peers can simply publish modifications as new resources, thus avoiding modifications at all.

However, in some applications the users need to work jointly on the same resource. This can be implemented using a consistency service dealing with concurrent modifications. There are two approaches to maintaining consistent data modification [54]:

1. The *pessimistic approach* requires a globally exclusive lock to be obtained before each update operation. This can be implemented using consensus protocols, like the Byzantine [58] or Paxos [57] algorithm. Etna [66] is an implementation of Paxos on DHTs. [10]

2. The *optimistic approach* allows for concurrent modification, and provides reconciliation in case resources became inconsistent, as implemented in the Concurrent Versioning System (CVS).

Application Example

In the collaborative product development scenario, business partners create quotation documents in a collaborative way. In the pilot implementation for the ATHENA project, only the originator of a document was allowed to modify it, which makes the requirement for concurrent modifications obsolete. However, in the general case, it would be possible that business partners modify documents concurrently, and consensus protocols needed to be used to avoid inconsistencies.

Impact Analysis

The figure below summarizes the effect of deploying a consensus protocol as a service layer component. The level of *reliability* is not affected by the use of consensus protocols, as they do neither introduce additional redundancy, nor do they decrease redundancy. In the previous section, replication was introduced as a means for meeting *limitations for network load*, applying load balancing among the peers in the replication group. Using consensus protocols,

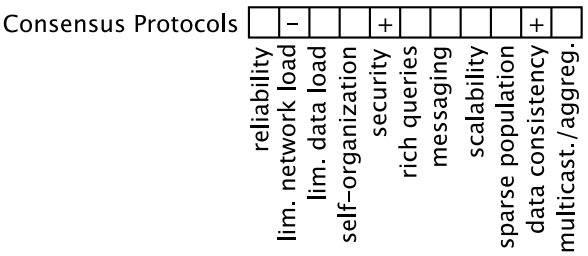
network load balancing becomes more difficult. While replication can be used to avoid that a single peer needs to reply all requests for a resource, consensus protocols have the effect that each peer in the replication group is consulted upon each request for a resource. Therefore, consensus protocols have a negative effect on requirements limiting the network load.

The *data load* is not affected by consensus protocols, as these protocols do not require a significant amount of data to be stored. Also, the *self-organization* is not affected, as the peers holding copies of a resource can run the consensus protocol autonomously, without the need for a central coordinator [10].

The plus on *security* is because using consensus protocols, a single peer can be kept from tampering with documents. This is especially true when using Byzantine agreements [58].

The support for *rich queries* and *messaging* is not affected by consensus protocols. The field for *scalability* is left blank, because consensus protocols slow the system down by a constant factor, independently of the size of the overlay [66].

The *sparse population* is not affected by consensus protocols. Obviously, consensus protocols provide *data consistency*. *Multicasting and aggregation* is not affected.



5.2.4. Public Key Infrastructure

Security was one of the requirements that played a role in all of the use cases presented in Section 4. A **public key infrastructure (PKI)** is a basic service layer component being a prerequisite for many security concepts. A PKI provides three core services [5, 95]:

- **Authentication**. The assurance that the originator of data resources or messages is who he claims to be.
- **Integrity**. The assurance that data resources or messages have not been altered intentionally or unintentionally.

- **Confidentiality**. The assurance that the contents of data resources or messages can be read only by a particular group of recipients.

Technology Description

Public key infrastructures are well-known building blocks in many IT security related applications. The most common standard for PKIs is X.509 [111]. PKIs require a trusted **Certificate Authority (CA)** signing the user's public keys. The signature can be verified by other users. That way, user can learn each other's identities. The certificates issued by the CA can be augmented with arbitrary information. For example, a peer identifier can be included in a certificate, binding peers with fixed positions in the overlay. In addition to that, most PKIs provide a certificate revocation list that is used for invalidating certificates.

Conceptually, the CA is a centralized entity, contrasting the decentralized architecture targeted in this thesis. However, it is not necessary that the CA is available at runtime. It is sufficient if the CA provides each peer with a certificate when the system is deployed. That way, the PKI can be used in a fully decentralized way at runtime.

When a peer-to-peer application is based on a VPN on the network layer, the PKI provided by the VPN can be reused in the higher layers, and only minimal service layer support is necessary. If the underlying layers do not provide security features, then a service layer component must be implemented providing access to the peer's public keys, the revocation list, and the CAs root certificate. Obviously, this data can be made available as resources in the DHT. This has been implemented at Siemens as part of the Peer Things project [92].

Application Example

In the collaborative product design scenario, business partners participating in a common business process may be both cooperators and competitors at the same time. These business processes are called **coopetitive** business processes [99]. There are two major security threats requiring that each peer is associated with exactly one, fixed identifier:

Firstly, having only one identifier per business partner guarantees that each partner operates only one peer. Otherwise, it would be possible for competitors to add sufficient peers to take control over large parts of the DHT, and to gain control over replication and routing of the business documents. This kind of attack is called Sybil attacks [31].

Secondly, having a fixed identifier prevents the business partners from choosing arbitrary positions in the DHT. To understand that second threat scenario, remember that the position in the DHT determines which resources

the peer is responsible for. If the position could be chosen arbitrarily, then a supplier could position its peer in a way that it becomes responsible for a certain kind of quotation documents.

Even if the contents of the quotes is encrypted, the supplier is still able to learn who of its competitors issued the quote, and to lower its price in reaction to this knowledge. If the positions in the DHT are bound with the PKI certificates, then choosing arbitrary positions in the DHT becomes impossible.

A more detailed security analysis of the CPD scenario is given in Section 6.

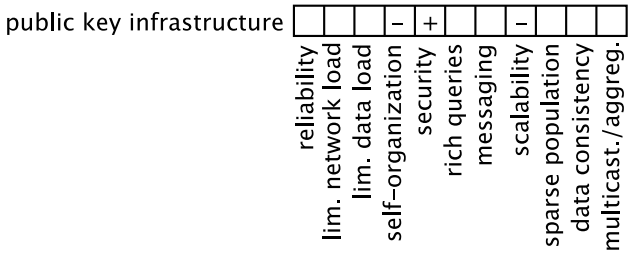
Impact Analysis

Implementing a service layer component providing a PKI is mainly motivated by security requirements, and it has only a small impact on other requirement areas. The *reliability* is not affected, neither is the requirement for *limited data load*. The CA's root certificate must be known to all peers, and the peer's public keys must be made available in the overlay, but storing a public key takes less than 2 kBytes[1]. Although requesting public keys requires additional network traffic, no negative impact on *limitations for network load* were observed in the Peer Things project, as public keys only need to be requested once, and can then be cached by the peers.

The negative effect on *self-organization* is due to the fact that the CA is a centralized instance, i.e. peers cannot participate in the PKI without being provided with a certificate by the CA. However, as said above, this effect is only relevant when deploying new peers, and does not affect the peers' runtime.

The plus on *security* is obvious. Support for *rich queries* and *messaging* is not influenced by a PKI component. The negative effect on *scalability* is due to the centralized nature of the CA, as an increasing number of users might result in a significant administrative overhead to issue the certificates. A survey of PKI and scalability issues can be found in [97].

The requirements for *sparse population*, *data consistency*, and *multicasting and aggregation* are not affected when introducing a PKI component.



[1]The size of an ASCII-encoded 2048 Bit public key generated with GnuPG is 1726 Bytes.

5.2.5. Caching

Like replication, caching techniques store data resources redundantly. But unlike replication, caching is not primarily used to increase reliability, but focuses on better query performance and reduced network load.

Technology Description

The idea behind caching is to exploit the recursive nature of most DHT routing algorithms. When querying data in a peer-to-peer overlay, a set of identifiers is queried, and the peers being associated with these identifiers are resolved. Recursive queries are forwarded from peer to peer, until a peer is reached that can reply.

The caching component causes the peers on the path to remember the results whenever they are involved in routing a query. There are different strategies to decide how long cached data should be remembered, and how much data should be cached per peer. An evaluation of these strategies can be found in the Diploma thesis by Michael Niebergall [69], which was conducted at Siemens Corporate Technology.

The most well-known application for caching in peer-to-peer overlays is Web caches, like Coral [41]. The idea is to relieve heavily loaded web servers by caching frequently queried content in a peer-to-peer overlay.

Application Example

Caching is an important component in the user directory scenario. One of the main problems with implementing a user directory is that some last names are queried very often (like Müller in a German phone book), while most last names are queried quite infrequently.

Section 7 will show how caching can be combined with a component providing range queries and enabling the implementation of a decentralized user directory based on a structured peer-to-peer overlay.

Impact Analysis

The figure below shows the impact of caching on the requirement areas identified in Section 4. As caching shares many properties with replication, most of the impact is the same as with the replication component.

As opposed to replication, caching has no positive effect on *reliability*, as caching does not aim at guaranteed access to backup resources. Rather, caching aims at increased *scalability*, which is not the focus of replication.

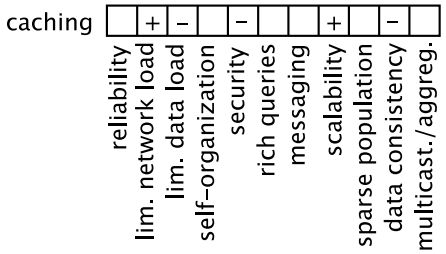
The other effects are similar to replication: The impact on *limitations for network load* and *limitations for data load* is a typical trade-off: Using caching increases the data load to be stored on the peer, while it reduces the network load when resolving queries. The parameters of the caching components must be chosen in a way resulting in a reasonable trade-off with respect to the use case scenario.

As with replication, *self-organization* is not affected by caching. The negative effect on *security* is a rather weak impact. There might be applications where it becomes a security issue when arbitrary peers on the path can answer arbitrary queries. This has to be investigated in detail for the given application scenario.

In the scenarios investigated in this thesis, no effect of caching on *rich query* support and *messaging* functionality has been observed. *Scalability* is increased, as caching reduces the number of peers involved when resolving a query. There could be a positive effect on the requirement for *sparse population*, but this was not observed in this thesis, and therefore this field is left blank.

Like replication, caching has a negative impact on *data consistency*, as modifications might not be performed as atomic transactions. However, unlike with replication there is not necessarily a fixed group of peers that cache data. Therefore it becomes harder to use consensus protocols to avoid the data consistency issues.

The *multicasting and aggregation* capabilities are independent of the use of caching.



5.2.6. Subscriptions

Subscriptions are needed if a peer requires to be notified when resources of interest are added, updated, or removed. Using subscriptions, it becomes possible to implement messaging and event-based applications on top of a peer-to-peer overlay.

On the plain overlay layer, this can only be implemented using polling, which would result in high network load. To avoid this, service layer components can be integrated to provide notification functionality.

Technology Description

The most simple approach is to use a special **subscription resource**, as implemented in the BRMF [93]. If a peer requires to be notified about resources with a given set of identifiers, it issues a subscription resource for each of these identifiers. The peer being responsible for an identifier receives this resource and learns that the subscribing peer must be informed upon events related to that identifier.

The disadvantage of this simple approach is that if n peers are subscribed for a certain keyword, the responsible peer must send n notifications upon each event. This causes significantly less network load as compared to using a polling mechanism, but for large n this is still not an ideal solution.

Reducing the network load is the focus of the dozens of multicast infrastructures that have been proposed for the dissemination of messages to a group of subscribers. Two of the earlier proposals were Scribe [20] for Pastry [19] and Bayeux [118] for Tapestry [117]. A survey of multicast solutions can be found in [1].

Application Example

Subscriptions are heavily used in the peer-to-peer based CPD scenario. Here, suppliers need to be notified if a manufacturer issues an RfQ document, and manufacturers need to be notified whenever a supplier issues a quote or a change request. As the number of subscribers for a given keyword is comparably low, it is feasible to use a simple service layer component based on subscription resources.

Impact Analysis

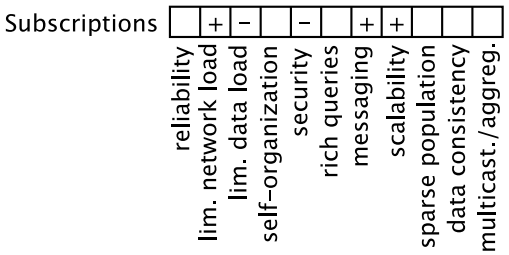
The effect of subscriptions on the requirements identified in Section 4 is summarized in the figure below. There is no effect on the *reliability*, as subscriptions are independent of the level of redundancy provided. As said above, subscriptions help to comply with *limitations for network load* in the business collaboration scenario, as they avoid polling. As a trade-off, *limitations for data load* might be violated, as the peers are required to store subscription resources or additional information for routing the notifications. The *self-organization* properties are not affected, as subscriptions can be implemented in a fully decentralized way.

The impact of subscriptions on *security* needs to be studied carefully with respect to the use case. For example, in case of the business collaboration scenario, subscriptions raise the risk that the peer being responsible for a

certain identifier learns who of its competitors is subscribed for that identifier. This issue will be discussed further in Section 6. Anonymous subscriptions have been proposed in [99].

Support for *rich queries* was not affected by subscriptions in the scenarios studied in this thesis. Obviously, subscriptions provide *messaging* functionality, which is their primary target. The level of *scalability* can be increased using subscriptions, as polling can be avoided.

There was no effect observed on *sparse population*, *data consistency*, and *multicasting and aggregation*.



5.2.7. Fuzzy Hashing

If a peer suffers from too much network traffic or too much data load, this peer is called a **hot spot**. Load balancing techniques are designed to avoid hot spots. The replication techniques introduced above provide a way to avoid hot spots in terms of network traffic. In this section, fuzzy hashing is introduced as a way to avoid hot spots in terms of data load.

Technology Description

Data hot spots occur if many data resources are associated with the same set of identifiers. In that case, all of these resources are stored on the same peer. The easiest way to avoid this is to choose better identifiers. Static analyzers create recommendations for good choices of identifiers based on a given sample set of resources [42].

In case that no alternative identifiers can be chosen, load balancing techniques allow for a better distribution of the resources. A very simple way to relieve hot spots is to use **virtual peers** [106]. The idea is that each physical peer runs more than one virtual peer in the overlay. If one of these virtual peers becomes a hot spot, the operator can turn off the other virtual peers to reduce the amount of data stored on the physical peer.

However, virtual peers are not effective if too many resources are stored on the same virtual peer, i.e. if too many resources share the same identifiers. In that case, load balancing techniques need to modify the hash function that maps the resource's keywords to identifiers. The original deterministic hash function is replaced with a probabilistic function. That way, resources with the same keyword are distributed among several peers.

There are several strategies for this kind of **fuzzy hashing**. The power of two choices [65] introduces random bits in the keyword. Other techniques, like thermal dissipation [88] make resources move away from hot spots and settle down on peers with less data load.

However, fuzzy hashing increases the number of queries that are necessary when a specific resource is searched. Without fuzzy hashing, only one peer needs to be queried for a certain keyword. With fuzzy hashing, more than one peer needs to be queried. Therefore, fuzzy hashing is only recommended for data that is accessed infrequently. Frequently used data should not be stored using fuzzy hashing.

Application Example

In the CPD scenario, RfQ documents are stored as data resources in a peer-to-peer-based decentralized collaboration platform. When the negotiation phase has finished, the documents may be archived in the collaboration platform for later reference.

There are only a few active documents that are currently negotiated. These documents are accessed very frequently, and strict hashing should be used for these documents in order to provide short access times.

However, there is a large number of documents where the negotiation phase is over. These documents are accessed very infrequently, and it is worthwhile to use fuzzy hashing when archiving these documents. That way, data load can be balanced when archiving these documents.

Impact Analysis

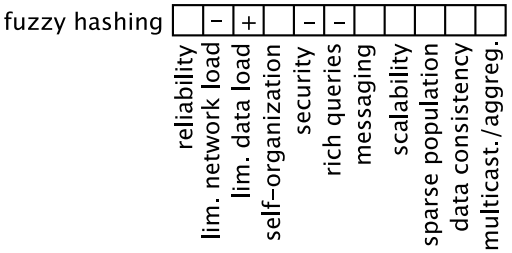
The effects of fuzzy hashing on the requirements is summarized in the figure below. There was no effect observed on *reliability*, as fuzzy hashing did not affect the level of redundancy. Basically, the use of fuzzy hashing requires a trade-off between *limitations for network load* and *limitations for data load*. The more data is balanced among different peers, the more peers need to be queried when searching for data.

The *self-organization* properties are not affected when using fuzzy hashing. Fuzzy hashing has a negative impact on certain *security* requirements. For

example, access control mechanisms in the business collaboration scenario would be violated if it is not clear which peer is responsible for a certain resource. On the other hand, subscriptions might even increase the level of security in other scenarios, as single peers can be kept from gaining the full control of all resources with certain identifiers. Security considerations will be discussed in Section 6.

The negative impact on *rich queries* is there because most rich query components require to know exactly where each resource is located in order to implement the query resolution strategy.

The other requirements, *messaging*, *scalability*, *sparse population*, *data consistency*, and *multicasting and aggregation* were not affected by fuzzy hashing in the scenarios considered in this thesis. Fuzzy hashing increases the time it takes to locate responsible peers, but the semantics of the mapping functionality remain the same.



5.2.8. Redundant Paths

One major security challenge in all DHT-based applications is to protect the DHT's routing mechanism from malicious modifications. In DHTs, queries are resolved by recursively or iteratively approaching the requested peer. If a peer on the routing path is operated by an attacker, it can maliciously redirect the queries. That way, many service layer components can be corrupted. For example, certain peers and resources can be made unavailable, and certain portions of the network traffic can be modified or mislead.

Technology Description

Redundant routing paths [98] are a way to prevent attacks on the DHT's routing. The idea is, instead of using a single path for resolving an identifier, several alternative paths are used. If each query is issued on several paths in parallel, a single malicious peer cannot influence the routing, because the other paths will still resolve the query correctly.

The easiest way to implement this is to maintain more than one DHT in parallel, and to store each resource in each of these DHTs. In case this causes too much network load, it is also possible to maintain several routing tables on top of a single DHT. That way, redundant routing is enabled, while the DHT's topology needs to be maintained only once.

Application Example

In the distributed user directory scenario, attackers could operate a modified peer to mislead queries for a certain name. For example, a certain name could be made unavailable in the user directory. If redundant routing paths are used, there is more than one path to resolve the query. Users can issue their queries several times in parallel on different paths, and the name is found in spite of the void paths.

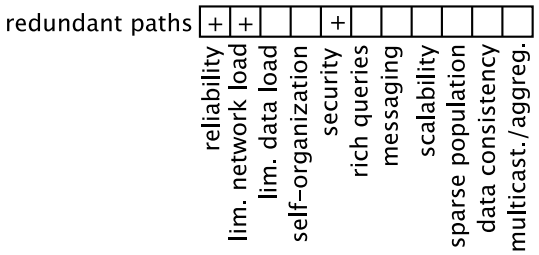
Impact Analysis

Using redundant paths increases the level of *reliability* of the application. Without redundant paths, the DHT's stabilization protocol detects peer failures, and adapts the routing tables accordingly. Redundant paths increase the level of reliability, as queries can be resolved even if the DHT's stabilization protocol fails temporarily.

Moreover, there is a positive effect on *limitations for network load*. While maintaining several routing paths causes an increased amount of traffic in the overall network, single peers suffering from peak traffic loads can be relieved, as alternative paths can be used for resolving queries. The requirement for limited network load in Section 4 refers to the maximum network load per peer, and not to the overall network load in the entire DHT. Therefore, redundant paths have a positive effect on this requirement.

The data load per peer is increased, because additional routing tables need to be stored. However, the amount of additional data is negligible, and there was no violation on any *limitation for data load* in the scenarios studied in this thesis. *Self-organization* is not affected, as the routing algorithms used in DHTs are fully decentralized.

The main purpose of redundant paths is to increase the level of *security*. For the other requirements, *rich query* support, *messaging* functionality, *scalability*, *sparse population*, *data consistency*, and *multicasting and aggregation* no effect was observed. The figure below summarizes the impact of redundant paths on the requirements identified in Section 4:



5.2.9. Search Index

The raw peer-to-peer layer offers a simple mapping of a set of identifiers to a set of peers. If an application does not require rich query functionality, then the identifiers can simply be hash values of keywords associated with the data to be stored. However, many applications require support for resolving more complex queries. In that case, service layer components must implement a search index that can be used to resolve complex queries.

Technology Description

When looking at query technologies in non-peer-to-peer-related work, four major trends can be identified. For each of these trends, attempts have been made to implement them in peer-to-peer-based applications. The following is an overview of related work in that area:

1. *Ontology* formats like OWL or RDF allow the applications to add semantic markup to the resources. This meta-data can be used to implement semantic queries. A peer-to-peer-based Ontology was successfully applied in the Edutella project [67].

2. *SQL* is the standard query language for database systems. Implementing a distributed SQL query processor on top of a peer-to-peer overlay is the focus of the Pier project at UC Berkeley [51].

3. *XPath* can be used to navigate in hierarchical XML data. Given a suitable XML Schema, a distributed XML tree can be created spanning over all resources in the peer-to-peer overlay. Distributed XPath queries have been investigated in the AXML project [3].

4. *Full text search* is of interest because it works in the absence of any markup or predefined structure of the business objects. Lucene[2] is a very

[2] http://lucene.apache.org, 2 August 2007

successful open source search engine library that can be used as a basis for implementing a full text search index. However, there is no related work on building a distributed Lucene index. Therefore, full text search is likely to require a central index server that processes search queries and returns the corresponding peer identifiers.

Application Example

In the CPD scenario, business partners need to search for RfQ documents being relevant for their business. In a simple implementation, this can be done without rich queries. For example, the name of the "part to be quoted" could be used as a keyword directly. However, this implies that all business partners choose their keywords from a common standard catalog of all product names, as the names must be unique and unambiguous.

Using a rich query component, the application becomes more flexible and more scalable. For example, an ontology of materials and manufacturing methods can be used to describe the parts to be quoted. That way, the business partners can query the RfQ documents using a semantic query language.

Impact Analysis

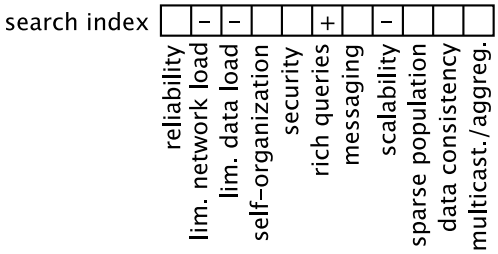
Although the related work on distributed ontologies, SQL, and XPath looks promising, research in that area is still far from generating results that are as efficient as their non-distributed counterparts. The main reason for this is that queries involving a large part of the overlay do never scale with an increasing number of peers. Therefore, distributed query languages will always be restricted to a subset of the non-distributed versions.

In Section 6 shows that state-of-the-art research in decentralized rich queries does not yet allow the implementation of a component supporting efficient range queries in the distributed user directory scenario. Therefore, a new component has been developed as part of this thesis, and it will be evaluated in Section 7.

The impact of rich query components on the requirement areas identified in Section 4 is shown in the figure below. There is no impact on *reliability*, as rich queries do not affect the level of redundancy in the system. *Limitations for network load* may be violated when a high number of peers is involved in resolving a query. *Limitations for data load* may be violated, because the indexing data must be stored in addition to the regular data load. The effects on network load and data load will be analyzed for the user directory scenario in Section 6.

The *self-organization* and *security* are not affected by rich queries in the scenarios studied in this thesis. Obviously, the requirement for *rich queries* can be fulfilled using a rich query component.

Messaging functionality is not affected. The level of *scalability* is likely to decrease when using rich queries, as often a large part of the peers need to be queried. There was no effect observed on *sparse population*, *data consistency*, and *multicasting and aggregation*.



5.3. Summary

In this section, related work was reviewed and ideas from related work were generalized as reusable service layer components. For each of these components, the underlying technology was described and an application example in one of the scenarios in Section 4 was given. The impact of the components on the requirement areas identified in Section 4 was described.

In the next section, a methodology is presented helping application developers to identify the gap between state-of-the-art in decentralized peer-to-peer computing and application requirements. The idea is to relate the components presented here to the use case requirements found in Section 4. It turns out that not all of the requirements can be fulfilled using the components presented above. Therefore, new service layer components need to be developed in order to implement the applications described in Section 4.

This methodology is applied to the application scenarios from Section 4, and new service layer components are implemented for each of the applications. As service layer components provide generic functionality, these new components can be reused by other developers in other application scenarios as well.

6. Applying a Methodology for Deriving New Components

6.1. Overview of the Methodology

This section presents a methodology helping application developers from industry to relate state-of-the-art research in decentralized peer-to-peer computing with their application requirements, and to identify the gap between current research results and the requirements.

The methodology is applied to the three industrial application scenarios presented in Section 4, and the missing functionality is identified. For each of the scenarios, a new service layer component is presented enabling the use of peer-to-peer-based decentralization for that application.

The methodology is generic, and can be applied in the same way to all three use case scenarios. The structure is as follows:

The scenario review revises the application scenario presented in Section 4, and splits required functionality into the different layers, according to the architecture introduced in Section 3.

The gap analysis relates the requirements found in Section 4 to the state-of-the-art presented in Section 5. As a result, the gap between state-of-the-art and the application requirements becomes clear.

The problem statement defines the functionality of the service layer components that need to be implemented to close that gap.

The technical description describes the implementation of the new component.

The related work compares the new components to state-of-the-art, and points out why it was necessary to develop new solutions.

The summary and outlook recapitulates the results and puts them in the context of current peer-to-peer research.

The goal of this section is to identify and close the gap between current research in decentralized peer-to-peer computing and the requirements of the application scenarios. An evaluation of the new components is presented in Section 7.

6.2. Confidentiality in Business Communication

6.2.1. Scenario Review

The first scenario introduced in Section 4 was the collaborative product design scenario, where suppliers from the automotive industry negotiate RfQ documents using a decentralized peer-to-peer-based business collaboration platform.

Network layer. On the network layer, a **virtual private network (VPN)** is used as the underlying network infrastructure. This prevents eavesdroppers from intercepting the peer-to-peer protocol data and ensures that no unsolicited participants join the network. The PKI provided by the VPN can be reused on the service layer to meet the security requirements.

Overlay layer. All business partners on the supplier side provide peers in a DHT. As shown in Section 2, the DHT approach provides guaranteed access to resources, while it does not depend on a central index server. These properties make DHTs suitable for decentralized business collaboration scenarios.

Service layer. The service layer components that need to be integrated are described in the gap analysis below. The aim of the service layer is to fulfill the requirements presented in Section 4, using the interface provided by the overlay. As a result, the service layer should provide a reliable collaboration platform enabling messaging functionality in a secure environment.

Application layer. The application focuses on the implementation of the business process steps taken in response to events on the collaboration platform. For example, the arrival of a new RfQ document triggers a quotation process on the supplier side. The deployment of the peer-to-peer overlay as well as the internals of the event handling are hidden to the application layer.

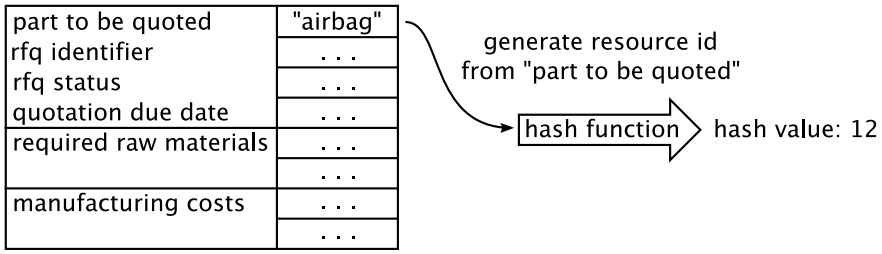


Figure 6.1.: Associating a Resource Identifier with a Quotation Document

6.2.2. Gap Analysis

The main benefit of the component-based service layer introduced in Section 3 is that it facilitates the analysis of the gap between state-of-the-art in peer-to-peer research and the requirements found in an application scenario. In the gap analysis, the requirements identified in Section 4 are reviewed, and the need for additional service layer components is identified.

Messaging is supported by a variety of current peer-to-peer applications, as presented in Section 5. In the pilot implementation of the CPD scenario, the BRMF's subscription mechanism is used [103], which relies on subscription resources.

Reliability is one of the major topics in peer-to-peer related research. In the implementation of the CPD scenario, reliability is achieved through replication. If a peer fails, the neighboring peers take over responsibility for the documents and subscriptions handled by that peer.

Self-organization is a low-level requirement that does not actually need service layer support. Rather, self-organization is provided by the underlying peer-to-peer layer. However, care must be taken if service layer components are used having negative impact on self-organization.

Limited network and data load are application requirements that must not be violated on the service layer. In Section 7, an evaluation of the pilot implementation shows that network and data load is well below the limits defined in the ATHENA project specifications.

Data consistency would usually require service layer components implementing consensus protocols, as shown in Section 5. However, in the pilot implementation of the CPD scenario, data inconsistencies are avoided by shaping the process in a way that concurrent modification of documents

becomes impossible. The business documents may only be modified by their originator. When more business partners work jointly on a specification, then both work on separate documents, and the documents are merged afterwards by the first tier supplier.

Rich queries require some kind of distributed search index, as described in Section 5. However, in the pilot implementation of the CPD scenario it was possible to use a simple, keyword-based approach for querying RfQ and quotation documents. As a keyword, the ''part to be quoted'' was used, which is included in each RfQ and quotation. An example is illustrated in Figure 6.1. All resources related to the keyword ''airbag'' are handled by the peer being responsible for identifier 12, because 12 is the hash value of the string ''airbag''. If a supplier wants to search for RfQs with the keyword ''airbag'', then the supplier needs to look up the peer being responsible for the identifier 12 on the overlay layer.

Security is a very generic term, addressing several requirements. In section 4, security was split into the three areas of confidentiality, integrity, and availability. Availability is provided by the replication mechanism, as described above. In order to guarantee integrity, a PKI component is used. That way, resources can be signed and encrypted, preventing them from being altered maliciously. The last point, confidentiality, cannot be guaranteed so easily. Although the contents of the resources can be encrypted, there is still a risk that the business partners learn who of their competitors issues quotations for a certain RfQ. Therefore, a new service layer component was developed as part of this thesis, providing confidential communication on top of a DHT.

The relation of the requirements with the service layer components presented in Section 5 shows that most requirements can be fulfilled using state-of-the-art research results. However, there is a gap between state-of-the-art and the application requirements, as confidential communication cannot be provided using components from related research. The remainder of this subsection describes the requirements for a service layer component providing confidential communication in a peer-to-peer-based decentralized business collaboration platform.

6.2.3. Problem Statement

The CPD scenario presented in Section 4 is implemented on top of a peer-to-peer-based, decentralized collaboration platform. On the service layer, subscription component is used to trigger events if documents of interest

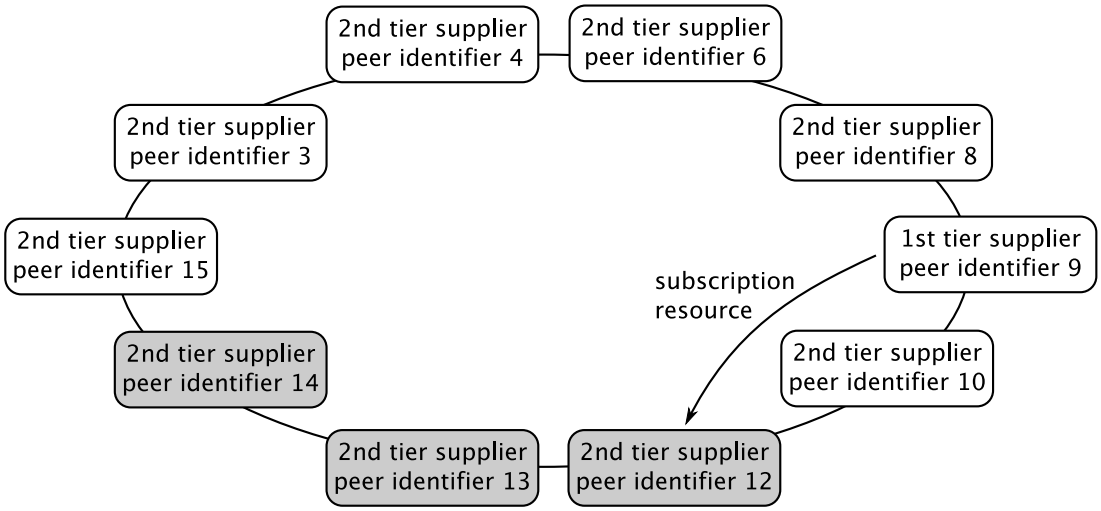


Figure 6.2.: Subscribing Events for Identifier 12

are created or modified. Figure 6.2 shows how the subscription mechanism works. The first tier supplier wants to subscribe for quotations with the keyword "airbag". As shown in Figure 6.1, these quotations are associated with the identifier 12, and are thus published on the peer being responsible for identifier 12. Therefore, the first tier supplier issues a subscription resource with identifier 12.

Figure 6.3 shows a second tier supplier publishing a quotation document for an airbag to be produced. The keyword "airbag" corresponds to identifier 12, and the quotation resource is thus stored on the peer with identifier 12 in the peer-to-peer overlay. The peers shaded in gray represent the replication group keeping backup copies of the original document.

The peer being responsible for identifier 12 checks its list of subscription resources and learns that the first tier supplier must be notified of the new quotation document. Therefore, peer 12 issues a notification, and the first tier supplier may in return request the new document from peer 12.

This means that all communication regarding airbags is routed via the peer being responsible for identifier 12. That peer might be operated by one of the supplier's competitors. Encryption can be used to make the content of the quotations only visible for the recipient. But even though the attacker cannot see the content of the quotations, two threat scenarios remain:

1. An attacker can observe whether a certain competitor places a bid, and lower the price in reaction to this knowledge.

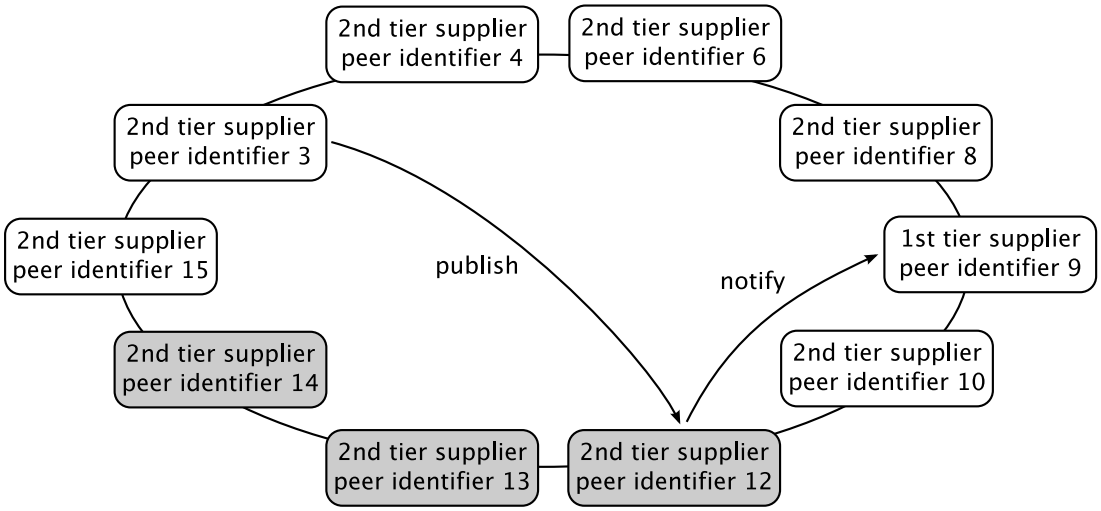


Figure 6.3.: Regular Publish Subscribe Mechanism

2. An attacker can keep track of its competitors and raise the price in case none of them places a bid.

In order to avoid this, there must be a service layer component providing means for publishing and retrieving objects confidentially. Sender authentication allows the receiver to learn who was the originator of a resource, but intermediate peers should not be able to tell where a resource comes from, and where it will finally go to.

As part of this thesis, a service layer component based on **onion routing** was developed. This component enables confidential communication on top of decentralized peer-to-peer overlays, and thus closes the gap between state-of-the-art and the requirements in the CPD scenario. The technical description below presents the implementation of this component.

6.2.4. Technical Description

The problem statement above motivated the need for a new service layer component enabling confidential communication. Two concrete threat scenarios were presented arising when a peer-to-peer based collaboration platform is used to implement the CPD scenario. In this subsection, the new component is described that can be used to meet the security challenges. The component is based on onion routing.

As a requirement, the confidential communication component depends on a

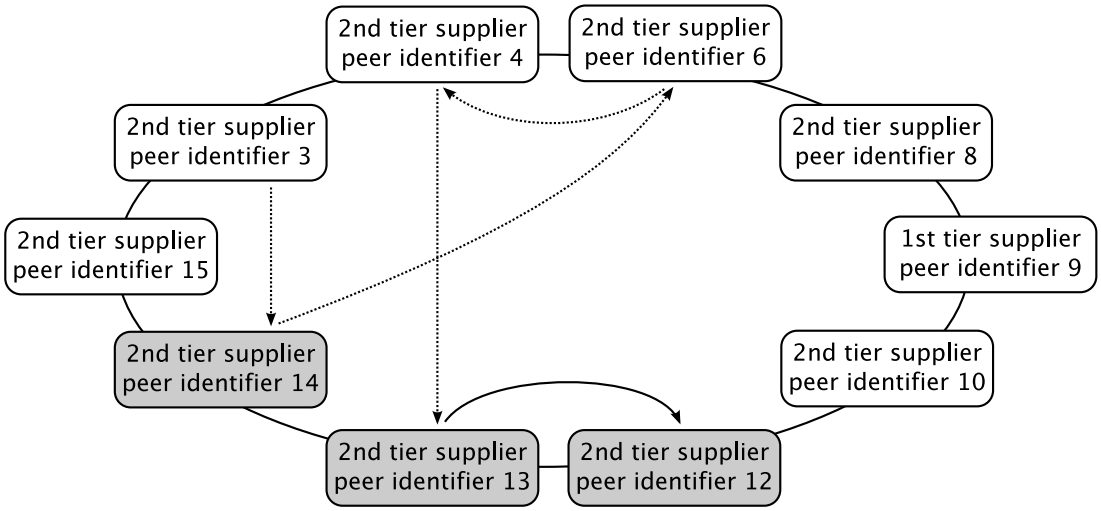


Figure 6.4.: Publish Operation Using Onion Routing

PKI component. Each peer must have one and only one signed public key. As described in Section 5, a lightweight component can be used importing the PKI from an underlying VPN, or alternatively a full PKI can be implemented on the service layer.

The basic idea of onion routing is illustrated in Figure 6.4. As in Figure 6.3, the peer with identifier 3 wants to publish a quotation associated with the identifier 12. Using onion routing, peer 3 does not contact peer 12 directly but chooses a random path through the peer-to-peer overlay. Then peer 3 encrypts the resource recursively as shown in Figure 6.5, and forwards the resource to the first hop on the path.

Each peer on the path decrypts the resource and forwards it to the next hop. The idea is that each node on the path just forwards some opaque, encrypted piece of information from one hop to the next, not being able to tell where the resource originally comes from and where it will finally go to. This is called onion routing, because the decryption is analogous to peeling the layers of skin from an onion. Finally, the last peer on the path publishes the resource. Note that the content of the resource may still be encrypted with the first tier supplier's public key.

The number of hops used for the onion path is a trade-off between performance, reliability, and security. As shown in the next section, the Mixmaster anonymous Remailer[1] uses a default path length of four hops, which is a rea-

[1] `http://mixmaster.sourceforge.net`, 11 September 2007

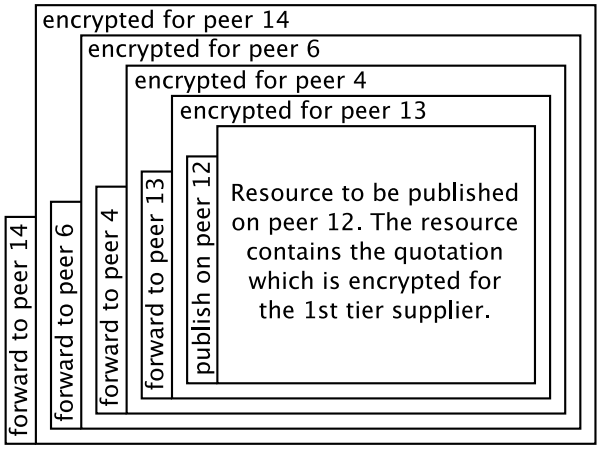


Figure 6.5.: Publish Operation Using Onion Routing

sonable value for our solution as well. In the prototypical implementation for the BRMF, a path length of 5 hops was used.

The identity of the original sender must not be known to the peers on the path. Therefore, it is important that the sender establishes the onion path silently, without interacting with the peers on the path. Therefore, peers continuously maintain a pool of peer locators and public keys in order to be able to build the onion path locally.

As a result, no peer on the path knows where a resource originally comes from, and where it will finally go to. Onion routing enables confidential communication among business partners, while the competitors are not able to tell who is talking to whom. Although the component was developed to solve a specific problem found in the CPD scenario described in Section 4, it can easily be applied in other scenarios as well, when confidential communication is required.

6.2.5. Related Work

While security in peer-to-peer networks is a relatively new topic (see [30] for an overview of current security issues in peer-to-peer networks), onion routing itself has existed for about 25 years. In this section, other applications of onion routing are reviewed and a discussion is presented about which features should be integrated in a business collaboration platform, and which features should be left out.

Originally developed for implementing anonymous email communication, onion routing nowadays is used in a large number of applications. Currently,

the most popular anonymous Remailer is Type II, as implemented by Lance Cottrell[2]. Type III Remailers [29] are still work in progress. Onion routing is also applied in the Java Anon Proxy[3], which is an HTTP proxy making it possible to surf the Internet anonymously. A more general approach is taken by Tor[4], implementing onion routing on the TCP layer, such that any network application can be anonymized by installing Tor on the underlying TCP layer.

In the peer-to-peer world, the traditional pragmatic focus is on anonymously publishing and downloading information, such as music and video files. Currently, rather sophisticated peer-to-peer systems implementing security features are available, like Gnunet[5], Freenet[22], and Crowds [85].

However, these systems lack the feature of notifications and do not provide guaranteed access to resources, which makes them less suited for implementing negotiation protocols than DHT-based solutions.

Looking at the existing implementations of onion routing, one frequently employed concept is the use of fixed-length messages, dummy traffic and random delays. That way, applications protect themselves against attackers who are able to trace the whole traffic in the entire network. As this is an unrealistic scenario in the CPD use case, this is left out in favor of simplicity, better performance, and less overhead.

Another interesting concept is circuits, as implemented in Tor, which has been mentioned above. When forwarding a message, a random identifier is sent together with the message. Each hop on the route maintains a table as shown in the table below:

incoming id	sender	outgoing id	receiver
⋮	⋮	⋮	⋮

Using that information, reply messages can be routed back along the onion path, while still no hop on the path can tell where the reply message originally comes from and where it will finally be sent to. It would be possible to extend this feature to implement anonymous subscriptions.

6.2.6. Summary and Outlook

In this subsection, the gap analysis was applied to the business collaboration scenario, and it turned out that confidential communication in structured peer-to-peer overlays has not been targeted by related work before.

[2] `http://mixmaster.sourceforge.net`, 18 February 2008
[3] JAP: the Java Anon Proxy, `http://anon.inf.tu-dresden.de`, 2 August 2007
[4] Tor: An anonymous Internet communication system, `http://tor.eff.org`, 2 August 2007
[5] `http://www.gnu.org/software/gnunet`, 2 August 2007

Therefore, a new service layer component was presented, that provides a confidential publish/subscribe mechanism. The component relies on onion routing, which has originally been developed for anonymous email communication. The onion routing algorithm was adapted to the DHT scenario, and DHT-specific implementations were introduced, e.g. a pool mapping peer identifiers to public keys.

The confidential communication component successfully eliminates the threat scenarios analyzed above, and thus enables the use of peer-to-peer-based decentralization in business collaboration in a secure way. The evaluation in Section 7 will show that the traffic overhead generated by the new component is well below the limits defined in the ATHENA project.

In the next subsection, the gap analysis will be applied to the distributed user directory scenario, identifying missing functionality that needs to be developed.

6.3. Search Index for the Distributed User Directory

6.3.1. Scenario Review

Above, the methodology of relating service layer components and application requirements was applied to the collaborative product design scenario. In this section, the same methodology is applied in the context of the distributed user directory application.

Network layer. The user directory is deployed on end user devices in the Internet. Therefore, reliability of the peers and performance of the network connections is not as high as in the CPD scenario above.

Overlay layer. The peers forming the DHT are implemented on telephones or DSL routers. Each peer knows its own entry in the user directory, and publishes that entry when it joins the overlay.

Service layer. The task of the service layer is to fulfill the application requirements identified in Section 4. The gap analysis below shows which components need to be integrated, and which are the yet unsolved problems.

Application layer. From the application's point of view, the user directory simply provides a query mechanism, where user entries can be searched. Wildcard-queries support users who do not know the exact data of the user to be looked up.

6.3.2. Gap Analysis

The gap analysis shows which requirements can be implemented using state-of-the-art technology, and which are yet unsolved challenges that need to be met with new service layer components.

Rich queries in the context of the distributed user directory means that it must be possible to perform range queries, like queries for all entries with a certain last name. There are ideas from related work that could be used to implement this as a service layer component. These ideas will be reviewed with the "related work" below. However, it turns out that none of them matches the requirements for scalability despite sparse population. Therefore, a new search index component needs to be developed.

Sparse population means that there is only one entry per peer in average, which results in a large number of peers storing no entry at all. Data density could be increased using a service layer component that distinguishes active and passive peers. However, the search index component presented here implements **skip list**s enabling scalable searches on top of a sparsely populated peer-to-peer overlay. Therefore, no additional service layer support is needed.

Scalability up to 500 000 peers requires caching components on the service layer to accelerate queries that are issued frequently. The search index must be implemented in a way that can be combined with caching components in order to meet the scalability requirements.

Self-organization is provided by the underlying peer-to-peer layer. The components on the service layer must be implemented in a way that does not counteract that property.

Reliability is achieved using a replication component on the service layer, as in the collaborative product design scenario. That way, even if a peer being responsible for an entry fails, neighboring peers take over the responsibility and the entry remains accessible.

Limited network and data load is not a requirement corresponding to a specific service layer component. Rather, it is a requirement regarding the overall combination of components on the service layer. The evaluation in Section 7 shows that the requirements from the Peer Things project are met when using the range query component presented here.

Security, as described in Section 4, can be divided into the three areas of confidentiality, integrity, and availability. While confidentiality doesn't

play a major role in a phone book-like application, and availability was already covered by the requirement for reliability above, providing integrity requires substantial service layer support. Establishing a PKI enables authentication of each peer. Including one and only one fixed peer identifier in the certificates eliminates the risks for certain attacks, as shown in Section 5. The PKI enables the peers to sign their entries in the user directory, protecting them from being altered maliciously. A detailed security concept was developed as part of the Peer Things project [92].

Reviewing the application requirements and state-of-the-art service layer components it was found that more research needs to be invested in range query support. In the next paragraph, the problem statement is given. Then, a service layer component enabling range queries in the user directory scenario is presented. The solution is compared to related work, and it turns out that none of the related approaches would have met the requirements of the distributed user directory. Finally, the summary and conclusions are presented.

6.3.3. Problem Statement

Decentralized communication platforms that enable Voice over IP telephony, video communication, and instant messaging are a prominent application of the peer-to-peer paradigm in the Internet. Many requirements found in communication platforms are either supported directly on the raw peer-to-peer overlay, or implemented by service layer components. However, addressing VoIP telephones requires the knowledge of a unique identifier or a unique phone number. Providing a distributed user directory where the unique identifier can be looked up in a decentralized way is yet an unsolved challenge.

The gap analysis above revealed that further research is needed to investigate in distributed search indexes enabling range queries, especially with respect to the requirement for scalability in spite of sparse population. The search index must allow **wildcard search**es. For example, a search for

```
OLPP*
```

should retrieve all entries where the last name starts with the string ‘‘Olpp’’. In case there are too many entries, the application should stop the query and ask the user to specify the query more precisely.

In the next paragraph, a service layer component is described providing range query functionality while complying with the requirements of the distributed phone book scenario. The following section on related work compares the component presented here with state-of-the-art research and shows why the range queries could not have been implemented using other search indexes.

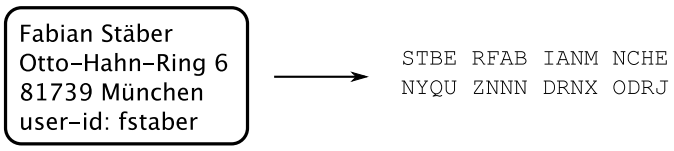


Figure 6.6.: Generating a 32 Char Identifier for an Entry

6.3.4. Technical Description

The **extended prefix hash tree (EPHT)** algorithm [101] presented here is a scalable indexing infrastructure supporting range queries on top of DHTs. The EPHT algorithm is based on the same idea as the Prefix Hash Tree (PHT) algorithm proposed in [82]. However, the EPHT differs from the PHT in several respects, which makes the EPHT applicable in the distributed user directory scenario.

In this paragraph, the building blocks of the EPHT algorithm are described. Starting from how to choose an identifier for a user entry, it is shown how trees are set up, and how range queries are performed using the trees. The performance is evaluated in Section 7.

Generating Identifiers

Each entry in the user directory is associated with a keyword. The keyword is a fixed-length string, consisting of the capital characters [A-Z]. Keywords are built by concatenating the data stored in the entry. In the example in Figure 6.6, the *last name*, *first name*, and *city* are concatenated to build the keyword.

The order of the concatenated data determines the relevance of the data for range queries. If keywords are built as in Figure 6.6, it is possible to search for the last name without knowing the city, but it is not possible to search for the city without knowing the last name. In order to allow alternative keyword orders, the application must maintain several trees in parallel.

Special characters like whitespaces or the German ä, ö, ü, ß are omitted. That way, both German names "Stäber" and "Stöber" map into the same string "STBER". It is up to the application layer to select the right results when a user searched for "Stäber".

The identifier length must be sufficient to ensure that a unique identifier can be built for each entry. In the evaluation in Section 7, the identifiers were 32 characters long. Identifiers that are longer than that are truncated, identifiers that are shorter are padded with random characters.

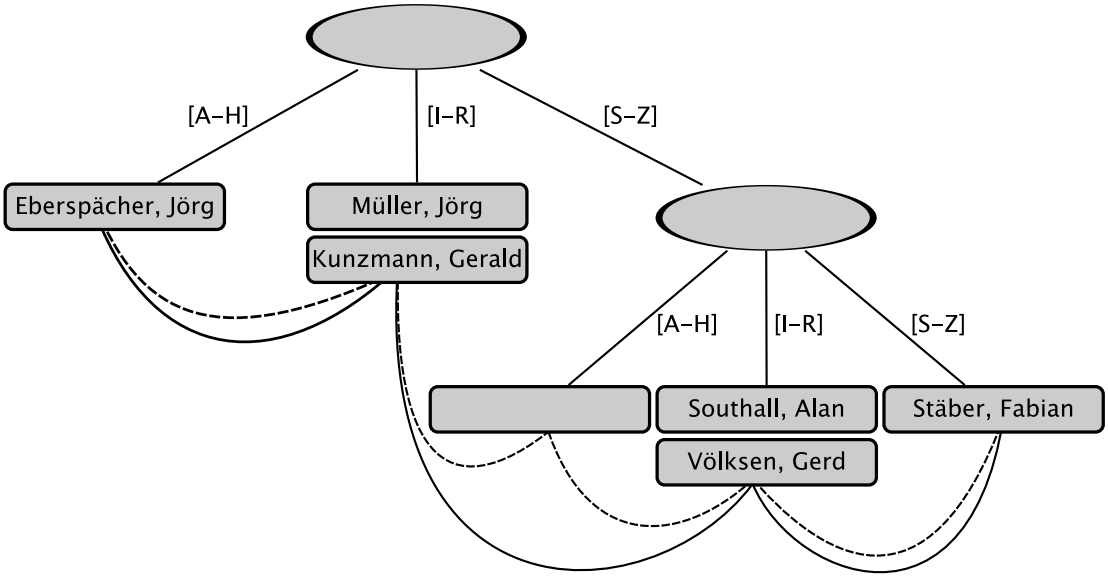


Figure 6.7.: An Extended Prefix Hash Tree with n = 3 and m = 2

Growing the Tree

The structure of an EPHT is shown in Figure 6.7. There are two parameters that determine the shape of the tree:

1. n is the number of children per node. Each edge is labeled with a character set, like [S–Z]. The partitioning of the alphabet into character sets is fixed and globally known, and cannot be changed dynamically during runtime. n is the number of character sets, which can be any number between 2 and 26. Figure 6.7 shows an EPHT with n = 3. The evaluation in Section 7 shows that the best performance is achieved with n = 26.

2. m is the maximum load of the root node, i.e. the maximum number of entries that can be stored on the root node. If the root node exceeds its maximum load, it splits up into n child nodes and distributes all entries among the children. The maximum load of each child equals the maximum load of the parent node plus one. The reason for incrementing the maximum load is to prevent recursive splits, if all entries happen to be stored on the same child. If a child node's *prefix* length (see below for the definition of *prefix*) equals the identifier length for the entries, then that node cannot split any further, and its maximum load becomes infinite. In Section 7 it is shown show that m = 100 is a good value.

Each tree node is stored as a data resource in the DHT. The resource identifiers are built using a hash function that operates in the **random oracle model** [12], i.e. even if two identifiers differ only in one single Bit, the hash values of these identifiers are two independent uniformly distributed random variables. The string to be hashed is the *prefix* of that node in the EPHT, i.e. the sequence of character sets on the path from the root node to the node to be stored. For example, the keyword of the leaf node holding the entry *Gerd Völksen* in Figure 6.7 would be `'[S-Z][I-R]'`. New entries are stored on the leaf node that has the closest matching prefix for the identifier of that entry.

Once a node is split, it becomes an inner node. Inner nodes are kept in the system to indicate the existence of child nodes, but they do not store any data. In particular, inner nodes do not need to store links to their children.

Maintaining the Linked Lists

In addition to the tree structure itself, two doubly linked lists are maintained: one for traversing the non-empty leaf nodes, and the other connecting all leaf nodes, including the empty ones. Each element in a list stores the prefix of its predecessor and successor. The linked list is updated upon the following events:

1. A leaf node splits up into child nodes. In that case, the old leaf node must leave the linked lists, the non-empty new child nodes must join the linked list for non-empty nodes, and all new leaf nodes must join the linked list connecting all leaf nodes. The new nodes learn about their successors and predecessors from their parent node.

2. An entry is added to a previously empty leaf node. In that case, that node must join the list for non-empty leaf nodes. The node finds its predecessor and successor using the list connecting all leaf nodes.

Performing Range Queries

Usually, tree algorithms imply that nodes are searched starting at the root node and traversing down the tree to a leaf node. This would mean that the peer holding the root node becomes a bottleneck and single point of failure in a distributed tree structure. EPHTs allow lookups to address arbitrary nodes directly, using the prefix of the node as the keyword in the DHT.

Range queries are implemented as follows: Firstly, the issuer of a query finds a random, non-empty leaf node lying somewhere in the queried range. Secondly, the issuer traverses the linked list of non-empty leaf nodes to the left

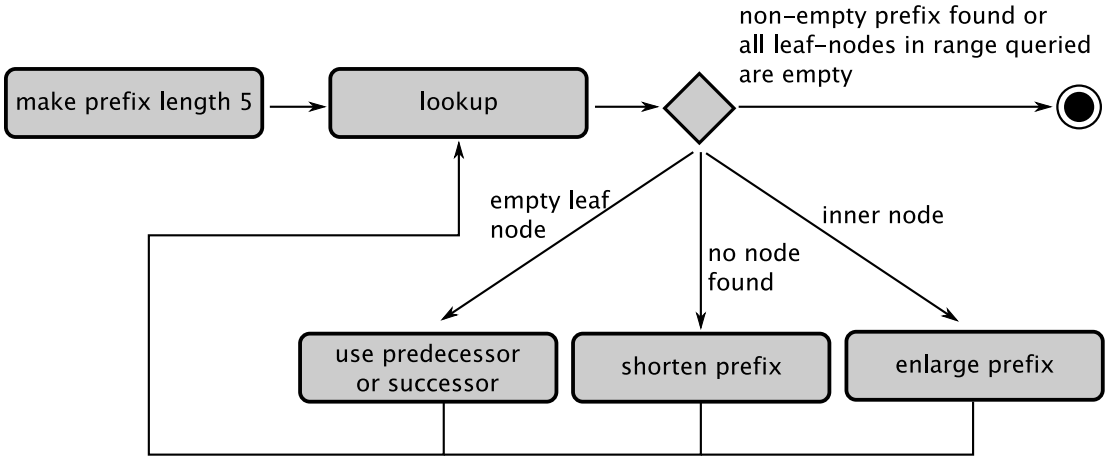


Figure 6.8.: Addressing Nodes

and to the right, subsequently querying the predecessors and successors, until all matching entries are retrieved.

Figure 6.8 shows how an initial non-empty leaf node is found that can be used as a starting point for traversing the linked list. To exemplify this, the following discussion shows the steps necessary when a user searches for all people with the last name *Kunzmann*.

- **Make Prefix Length 5.** The first step is to pad the search string with random characters, and to take the first five characters as an initial prefix to start with. In the example, the initial prefix would be `KUNZM`. The evaluation in Section 7 shows why `5` is a good initial prefix length.
- **Lookup.** When this prefix is looked up in the DHT, there are four possible results:
  1. A node with that prefix exists and is a non-empty leaf node. In that case, the initial node for traversing the linked list is found.
  2. A node with that prefix exists and is an empty leaf node. In that case, the issuer of the query starts traversing the linked list until a non-empty member is found. If all prefixes in the range queried are empty, then the search was unsuccessful.
  3. A node with that prefix exists but is an inner node. In that case, the prefix was underspecified, and it must be enlarged by one character. In the example, the next search string might be `KUNZMA`. If the search string was too short, it is padded with a random character.

4. There is no node with that prefix. This means the prefix was over-specified, and it must be shortened by one character. In the example, the shortened prefix would be `KUNZ`.

In order to decrease latency, the search can be initialized with several different random paddings in parallel. That way, the linked list can be traversed starting from different positions at the same time.

Removing Entries

Entries do not need to be deleted explicitly. Each entry is associated with a lease time. If it is not renewed within that time, it is deleted. That way, users who are absent for some time cannot be called.

In EPHTs, once a node has split and become an inner node, this node stays an inner node for ever, even if all entries in its sub-tree have timed-out. That means that the EPHT can only grow, but never shrink. This property is in accordance with the use case, as shrinking the tree would make sense only if the service provider operating the distributed phone book application would permanently loose a significant number of customers, or if the distribution of the name's prefixes changes significantly. Both scenarios proceed very slowly, and it is feasible to roll out a software update in that case that will built a new tree from scratch. The persistence of inner nodes enables the implementation of extensive caching.

Caching

As the EPHT never shrinks, inner nodes are immutable. They will never be deleted or altered. That means that inner nodes can be cached infinitely in the DHT. Caching components have been introduced in Section 5. Whenever a peer learns about the existence of an inner node, it can cache that information and respond when that prefix is queried the next time. Without caching, prefixes that are accessed very frequently would cause a lot of network traffic for the peer being responsible for that prefix. Using caching, this network traffic can be balanced in the DHT.

Comparison with the Original Prefix Hash Trees

The EPHT algorithm presented here derives from the Prefix Hash Tree (PHT) algorithm proposed in [82]. However, changes were introduced to the original PHT in order to implement the distributed user directory. The contribution of the work presented in this thesis is twofold:

1. The original PHT is a binary tree enabling Bit-wise processing of keywords. It is designed in a way that the combination with a caching component is impossible. Multiple keywords are handled using a Squid-like approach, which does not scale with popular last names, as described with the related work below.

   Therefore, the requirements found in the distributed phone book scenario cannot be met with the original PHT algorithm. The EPHT extends the PHT in several respects, as described below.

2. Unlike the original PHT, the EPHT algorithm has several configuration parameters, like the number of child nodes n, and the maximum load of the root node m. The EPHT is evaluated with real-world phone book data in Section 7. It is shown how to gain the best performance in the phone book scenario.

In the remainder of this section, the major differences between the EPHT and the PHT algorithm are presented.

- The original PHT is a binary tree. As shown in Section 7, binary trees do not scale well in the user directory scenario. Therefore, the EPHT is an n-ary tree, and the evaluation in Section 7 shows that best performance is achieved with $n = 26$.

- In the original PHT, if the number of entries in a subtree falls below a certain threshold, that subtree collapses into a single leaf node. The EPHT can only grow, but never shrink, which enables the combination with caching components on the service layer.

- Empty nodes are not handled specially in the PHT algorithm. In the EPHT, an additional linked list is introduced, skipping the empty nodes to improve performance. This list is added in order to make the EPHT perform better with sparse population. A significant number of prefixes do never appear in user's names, which results in empty leaf nodes for these prefixes.

- The original PHT proposes to handle multiple keywords using locality-preserving hashing, as in Squid. In an EPHT based application, keywords are simply concatenated according to their priority, and the result is padded with random data.

Algorithm Summary

In this subsection, the EPHT algorithm was presented that can be used to implement a service layer component providing a search index for range queries in the distributed phone book scenario. The presentation was divided into algorithmic building blocks, like the algorithm to build identifiers, the algorithm to split nodes when they reached their maximum load, the algorithm for resolving queries, and the algorithm for maintaining the linked lists.

The EPHT algorithm is based on similar ideas as the PHT algorithm. A comparison of the two algorithms was presented, highlighting the novel ideas introduced with the EPHT. In the following section, the EPHT is compared to related work, showing why the implementation of range queries was not possible using current state-of-the-art approaches.

6.3.5. Related Work

Above, the EPHT algorithm was presented that can be used to implement a new service layer component providing a search index for range queries in the user directory scenario. The implementation of range queries in peer-to-peer overlays has been a research topic before, and several solutions have been introduced in related publications. In this section, related work is investigated further, and it is shown why the requirements found in the user directory scenario cannot be implemented with related approaches.

When entries are stored in a DHT, the location of an entry is defined by its identifier. A common way to achieve a uniform distribution of the entries among the peers in the DHT is to generate the identifier using a hash value of the entry's name. The hash function, like MD5 or SHA1, operates in the random oracle model [12], i.e. even if two identifiers differ only in a single Bit, the hash values of these identifiers are two independent uniformly distributed random variables.

While these hash functions allow for good balancing of the data load in a DHT, they make range queries very costly. Iterating among a range of identifiers that are lexicographically next to each other means addressing peers in a random order in the peer-to-peer network. A way to accelerate range queries is to abandon the random oracle model, and to store the entries in lexicographical order. In the following, three approaches are introduced relying on this idea: Squid [94] and Skip Graphs [7], and Mercury [13].

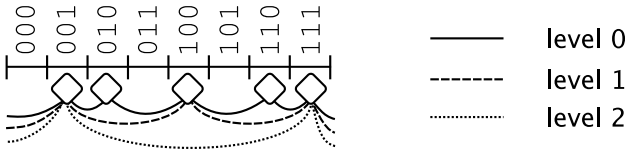


Figure 6.9.: Skip Graph

Skip Graphs

Figure 6.9 shows a linear 3-Bit identifier space. The peers, as being indicated by diamonds, are randomly distributed among the identifiers. Each peer is responsible for the identifiers in the range between itself and its predecessor or successor.

As shown in Figure 6.9, Skip Graphs introduce several levels of linked lists for traversing the peers. The higher the level of the list, the more peers are skipped, accelerating routing to specific ranges. By maintaining several independent lists on each level in parallel, Skip Graphs provide balancing of the traffic load and resilience to node failure.

However, the problem with Skip Graphs is that the entries' identifiers are not distributed uniformly among the linked list. Entries for a last name starting with S are very common in the German phone book, while last names starting with Y are very uncommon. Therefore, the peer being responsible for a common entry becomes a hot spot in terms of network traffic and data load.

This effect was evaluated by Gerald Kunzmann in a cooperation project with Siemens and the Technische Universität München [55]. The results will be part of Gerald Kunzmann's dissertation thesis.

Squid

Squid [94] is an approach for combining several identifiers when determining the position of an entry in the DHT. Squid is based on **locality-preserving hashing** [52], in which adjacent points in a multi-dimensional domain are mapped to nearly-adjacent points in a one-dimensional range.

For example, in a distributed phone book application, one could use a two-dimensional identifier space, where one dimension is the entries' last name, and the other dimension is the entries' first name. Figure 6.10 shows how two dimensions can be mapped on a one-dimensional range using a **space filling curve** (SFC). The SFC passes each combination of the two identifiers exactly once. If the user wants to search for all entries with a last name starting with ST and a first name starting with F, then the user simply needs to query the

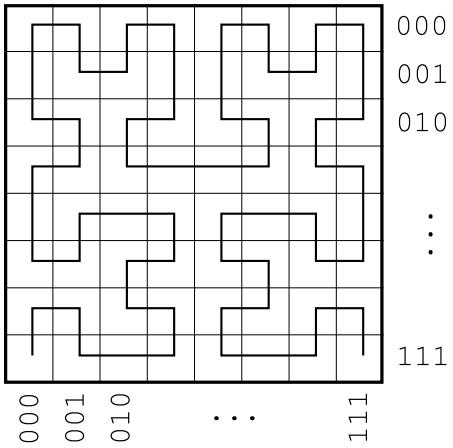


Figure 6.10.: Squid

parts of the range that lie on the intersection of these two prefixes in the two-dimensional space.

However, as with Skip Graphs, it turns out that the distribution of names in a phone book results in combinations that are very common, while other combinations are very rare. Again, the peers being responsible for common combinations become hot spots in terms of data storage and traffic load. This could be avoided with Squid by introducing many dimensions in order to distribute the entries among many different peers. But introducing many dimensions results in a tangled-up SFC. As a result, many short fragments of the curve need to be processed for each keyword that is not specified in a query.

Squid was evaluated as a promising candidate to be implemented as part of the Peer Things project. It turned out that in average a very high number of peers needs to be queried in order to find an entry, which results in high latency and network traffic. Moreover, hotspots were observed, violating the limitations for the data load in the Peer Things project. The EPHT presented above solves this problem by defining a threshold value for the maximum number of entries to be stored in a tree node.

Mercury

Like Squid, the Mercury [13] approach supports multi-dimensional keywords. Each dimension is handled within a separate hub, which is a ring-shaped topology of peers. An example of a Mercury hub is illustrated in Figure 6.11.

The ID range within a hub is ordered linearly, which results in the same load

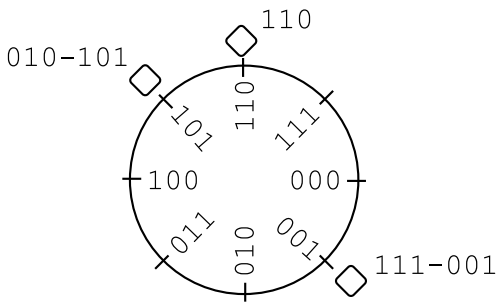


Figure 6.11.: Mercury

balancing problems as with the other approaches. However, Mercury suggests that peers are moved around dynamically to balance the load. Although this might be a reasonable approach in other scenarios, this raises difficulties in the user directory scenario.

Firstly, there are a few very popular last names. A peer being responsible for one of these popular last names cannot be relieved by moving around other peers, and it will stay a hot spot in terms of data load. Secondly, if peers may choose their position in the overlay deliberately, this raises certain security issues, because an attacker who wants to make a person unreachable can position its peer in a way that it becomes responsible for routing queries to the victims entry.

Summary

The brief survey of related work showed that there are several difficulties with previous range query solutions when applied to the distributed user directory scenario. A more detailed overview of search methods in peer-to-peer systems has been published in [90].

Approaches supporting real multi-dimensional keywords like Mercury and Squid have the problem of hot spots with very popular last names. Additionally, approaches relying on linear keywords instead of real hashing suffer from overloaded peers being responsible for popular prefixes. In Squid, the hot spots in terms of data load could be avoided, but as a trade-off this results in a large number of peers to be queried to find an entry.

As evaluated in Section 7, the EPHT algorithm above does not suffer from these problems in the user directory scenario. EPHTs enable efficient range queries, while preserving the advantages of the random oracle model for hashing, which results in a balanced distribution of the entries among the peers in the DHT.

6.3.6. Summary and Outlook

In this subsection, the service layer methodology was applied to the distributed user directory scenario in order to identify which of the application requirements can be solved using state-of-the-art technology, and which ones require new service layer components. Most requirements can be fulfilled using service layer components presented in Section 5.

Although significant research on rich query support is available, none of the existing approaches fulfills the requirements for performing range queries in a distributed user directory. Within the Peer Things project, Mercury, Squid, and Skip Graphs were evaluated, but it turned out that none of these approaches can be used to store and query the Zipf-distributed user data efficiently.

Therefore, a new service layer component was developed as part of this thesis that enables range queries on top of DHTs. Like the prefix hash tree, the component presented in this subsection implements a tree structure on top of a DHT. However, the range query component differs from the PHT in several ways, e.g. it is n-ary instead of binary, it uses concatenation of multi-dimensional keywords instead of space filling curves, and it can be combined with a caching component.

As the result, all building blocks for the implementation of the user directory are now available. The evaluation in Section 7 will show that the range query component keeps the maximum data load below the limits defined in the Peer Things project. That way, the implementation of a fully decentralized searchable user directory becomes possible.

In the next subsection, the service layer methodology will be applied to the distributed power generation scenario, and the gap between state-of-the-art functionality and the application requirements will be identified.

6.4. Routing and Aggregation Infrastructure for Distributed Power Generation

6.4.1. Scenario Review

The third application scenario introduced in Section 4 is a reliable control infrastructure for distributed power generation [100]. The control infrastructure establishes tree-shaped views, in which the power generating devices are situated in the leaf nodes. The tree can be used to query data from the power generators, and to send control signals to the power generators.

In the following, the application is separated into the different layers according to the architecture presented in Section 3:

Network layer. As the infrastructure should be able to connect a large share of households in the future, it is not feasible to require new communication hardware to be set up for each participant. Rather, the ubiquitous Internet is used to connect the utility and the households.

Overlay layer. The required scalability is achieved by establishing a DHT on top of the network layer. All peers forming the DHT are operated by the utility. In order to increase reliability, the peers should be located in different data centers.

Service layer. Internet connections are potentially unreliable, and the peers building the DHT might fail or become temporarily unavailable. Therefore, a reliable multicasting- and aggregation infrastructure must be implemented on the service layer, providing resilient routing of sensor and control data. The infrastructure must be complemented with other service layer components, e.g. components fulfilling the security requirements.

Application layer. From the utility's point of view, the power generators are logically located at the leaf nodes of a tree structure. The underlying physical location of the peers is not visible to the application. The tree structures are resilient to node failure, and can be used to send multicast messages to the power generating devices, and to aggregate sensor data received from these devices.

6.4.2. Gap Analysis

As with the other use case scenarios before, the methodology derived from the architecture in Section 3 is used to derive the service layer components needed to implement the control infrastructure application. The gap analysis relates the application requirements with the service layer components described in Section 5, and yields the gap between state-of-the-art research and the application requirements. This gap is closed with the new service layer component introduced in the remainder of this section.

Reliability is implemented using a replication component on the service layer. Resources are used to indicate that a peer is responsible for a node in the aggregation tree. If the peer fails, another peer from the replication group becomes responsible for the resource, and takes over the routing functionality.

Messaging can be implemented in a similar way as the subscription mechanism in the collaborative product design scenario. Resources are used

to indicate that a peer is responsible for forwarding messages. However, unlike the simple subscription mechanism in the CPD scenario, messaging in the control infrastructure requires multicasting and aggregation.

Multicasting and aggregation of sensor data is a new topic in peer-to-peer related research. In this section, a service layer component will be presented implementing a multicasting and aggregation tree on top of a peer-to-peer overlay.

Limited network load is supported by uncoupling the aggregation tree and the underlying hardware. If the network load exceeds the capabilities of the peers, then new peers can be added, and the network load distributes uniformly among all peers.

Scalability and self-organization are provided by the underlying peer-to-peer layer. The service layer components must be combined in a way that does not counteract these properties.

Security, as with the other scenarios presented before, requires a component providing a PKI. In the CPD scenario, peers operated by competitors joined the same peer-to-peer overlay, which required a complex security concept. In the distributed power generation scenario, the entire peer-to-peer overlay is operated by a single utility, which reduces the security implications. The main task of the security components is to provide sender authentication and encryption, in order to keep attackers from joining the peer-to-peer overlay, and to keep the sensor data confidential. Security implications of the multicasting and aggregation component are surveyed below.

6.4.3. Problem Statement

The gap analysis above showed that most requirements are supported by service layer components presented in Section 5, using state-of-the-art ideas from peer-to-peer research. However, the routing and aggregation functionality was not found in related work, and therefore a corresponding service layer component needs to be implemented to provide routing and aggregation infrastructure on top of a peer-to-peer overlay.

In the remainder of this section, a routing and aggregation component is presented. The component is designed in a way that supports the requirements for scalability and limited network load.

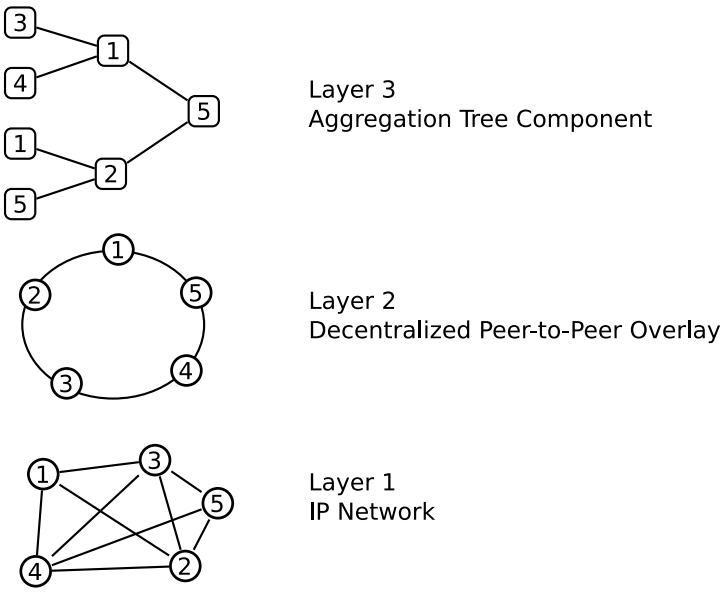


Figure 6.12.: Layered Architecture Applied to Aggregation Tree Component

6.4.4. Technical Description

The technical description explains the tasks performed by each peer, and it points out how the tree structure is preserved in case of node failure. A measurement of the reliability is given in Section 7.

Layers

Figure 6.12 shows the layered architecture from Section 3, as applied to the routing and aggregation component. The routing and aggregation tree is deployed on top of the peer-to-peer overlay. The tree structure is independent of the underlying hardware, which is a prerequisite for the scalability of the application.

1. On the IP layer there are hosts operated by the utility. The hosts are located in different data centers, and are connected through the Internet. In the example in Figure 6.12, there are five hosts.
2. On the overlay layer, each host provides a peer in a DHT. The number of peers equals the number of physical hosts on the IP layer.
3. The nodes in the aggregation tree are associated with data resources stored in the DHT. Each peer may be responsible for one or more resources. The

tree structure may consist of many more nodes than the underlying DHT. For example, in Figure 6.12 two tree nodes are stored on peer number 1. A replication component provides resilience: If a peer fails, another peer from the replication group takes over responsibility for the data resource, and takes thus over traffic for the corresponding nodes in the tree structure.

The application shows one of the major benefits of the layered architecture introduced in Section 3: The control infrastructure is set up independently of the underlying hardware, and provides many more tree nodes than physical hosts are available. The stability and scalability of the peer-to-peer overlay is used, while the topology of the overlay is hidden to the application layer.

Client Model and Security

In the control infrastructure, tasks are distributed between the utility and the power generators. The distribution of responsibilities allows to circumvent certain security risks that would otherwise have impact on the reliability of the control infrastructure.

All peers building the DHT are operated by the utility. The power generators provide servers at the leaves of the control infrastructure that deliver data and respond to queries, but these servers are not used for routing queries to other power generators. The architecture is illustrated in Figure 6.13.

As the entire DHT is operated by the utility itself, it is easily possible to run all the peers within a trusted VPN, where access from outside is prohibited. That way, it can be assumed that all peers in the DHT are trusted nodes, which decreases the risk of security threats on the DHT protocol layer. It is not possible for an attacker to send forged routing messages or to establish faked peers unless the attacker knows the utility's private key.

The utility itself acts like a client that can issue queries and will receive responses. The queries are signed by the utility, and the peers can verify the signature using the utility's public key that is used in the VPN. Refusing to route unsigned messages reduces the risk of denial of service attacks.

The routing infrastructure provides a virtual tree structure that is physically distributed among the peers operated by the utility and the servers operated by the power generators. The routing algorithm needs to know which part of the tree can be accessed on the peer-to-peer overlay, and which part lies on the power generator's servers. This separation is implemented using namespaces. The parts of the tree that are located on the utility's peers are under the namespace of the utility, while the subtrees lying on a power generator's server are under the namespace of the power generator.

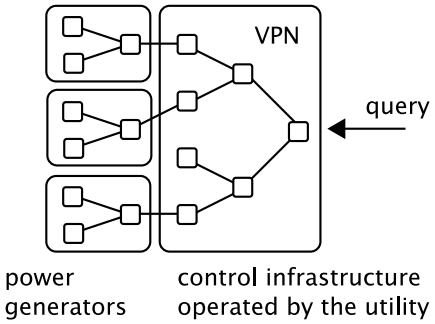


Figure 6.13.: Client Model

Power generators may define their own data structures under their namespace at the leaf of the tree. However, only the utility's part of the tree is deployed on top of the peer-to-peer overlay, and can thus benefit from the reliability and scalability. Therefore the proprietary data structures of the power generators should be kept as small as possible, and all data structures that are used by more than one power generator should be defined under the utility's namespace.

Expressing Paths and Queries

The routing and aggregation component in combination with a replication component enables routing of queries and responses in a reliable way, i.e. the responses reach their destination even if a peer on the path fails after the query was sent. The following explains how queries are resolved in the routing and aggregation component. As the first step, a definition is given of how paths and queries are expressed syntactically. For simplicity reasons, namespace prefixes are omitted in the remainder of this thesis, and the focus is on the utility's part of the tree that is distributed and replicated in the DHT.

The syntax of path and query expressions is taken from the XPath specification, and similar expressions have been used in related work on tree structures on top of peer-to-peer overlays. The similarities and differences between the routing and aggregation component and related work is presented below.

The tree nodes of the control infrastructure are stored as data resources in a DHT. Each resource is associated with a unique keyword. A hash value of that keyword is used as the identifier to look up the data resource in the DHT. As the keyword, the unique path expression is used that is associated with each node.

A **path expression** is a unique and unambiguous notation identifying a node

in the tree structure. With regular XPath, there are different ways to express a path to the same node. In the control infrastructure, a non-ambiguous way is needed to construct a path expression for each node in order to get a one-to-one mapping between the nodes and the corresponding paths. Each node in the distributed tree structure is required to have an `id` attribute, and the value of this attribute must be unique among the node's siblings. The path expressions are built using the name and the id of each node as follows:

$$/node_1\texttt{[id='}id_1\texttt{']}/node_2\texttt{[id='}id_2\texttt{']}/...$$

Query expressions are used when querying data in the tree structure. Unlike path expressions, query expressions are not required to be unique, i.e. it may be possible that there are different ways to address the same data.

There are three kinds of steps that can be combined to form a query expression:

Singlecasts have the form `/node[id='...']`. They are used for navigating the child axis of the tree. As the `id` is unique among a node's siblings, these expressions address exactly one child.

Multicasts are used for addressing multiple children. There are three types of multicasts. The first type is similar to a singlecast, but uses an other attribute than `id`, like `/node[type='...']`. As the `type` attribute is not required to be unique, this can be used to address multiple children. Secondly, if the attribute is omitted, as in `/node`, the step does address all children that are named `node`. The third type is `/*`, and it can be used to address all children regardless of their name.

Broadcasts have the form `//` and are used for broadcasting a query to an entire sub-tree.

As an example, the tail of the expression

$$...\underbrace{\texttt{/generator[type='biomass']}}_{\text{multicast}}\underbrace{\texttt{/status[id='c']}}_{\text{singlecast}}\underbrace{\texttt{//}}_{\text{broadcast}}$$

is a valid query expression, consisting of a *multicast*, a *singlecast* and a *broadcast*. The expression could be used to query the capacity data of all biomass generators in a certain region[6]. As with path expressions, the syntax of query expressions is taken from the XPath specification, but only a small set of XPath expressions are valid query expressions as defined here.

[6] In the example, the node `<status id="c">` has the capacity data as its child nodes.

Query Types

There are three query types that are supported by the control infrastructure: *read*, *write*, and *schedule*. In the following these query types are introduced, and it is shown how they are treated in the control infrastructure.

Write queries can be used to send control messages to power generators. For example, the utility could issue a write query to tell a biomass energy generator to increase its energy output. The response message for write queries contains the power generator's data after the update operation. The utility learns about unsuccessful updates by comparing the data in the response message with the expected values.

Read queries are used to read data from one or more power generator. Read queries are implemented as a special case of write queries, where the value to be written is *nil*. That way, the routing infrastructure does not need to distinguish between read and write queries.

Schedule queries are used to schedule periodic reports. For example, if the utility wants to stay updated with the current capacity of biomass energy generators, it schedules a periodic report, requesting the power generators to submit their capacity in a specified time interval. The control infrastructure handles schedule queries in the same way as read queries, except that there is not a single response message, but responses are repeated periodically.

The next paragraphs show how these queries and the corresponding responses are routed, and how the data in the response messages is aggregated.

Resolving Queries

In this paragraph, the algorithm for routing queries from the utility to the power generators is presented. The approach presented here shares similarities with related work on distributed XML presented below. However, the approach presented here provides some novel properties:

Firstly, the algorithm presented here is *stateless*, i.e. all data needed to handle a message is included in the message itself. If a peer fails after routing a query, any other peer is able to take over and route the response. This facilitates the composition of the routing service and a replication service, and allows the implementation of applications with a high level of reliability.

Secondly, the algorithm presented here is *efficient*, using a chunk-based routing approach that allows us to skip nodes in the tree, and to relieve the root node of the tree.

Thirdly, route optimization is provided using *locality* properties of the tree structure.

Routing of Queries using Chunks. In order to resolve a query expression, the relevant part of the tree structure needs to be traversed. Regular tree traversal algorithms navigate a tree node by node, starting at the root node. The main benefit of using a DHT as the underlying peer-to-peer infrastructure is that tree nodes can be accessed directly using the nodes' path expressions, i.e. knowing the path expression one can "jump" directly to any position in the tree. The chunk-based routing algorithm presented here exploits this, and provides efficient routing.

Before a query expression is processed, it is divided into the head that is under the namespace of the utility running the tree, and the tail addressing data in a power generators' servers. The remainder of this section only shows how to resolve the head of the query. The interaction with the power generator's servers is opaque to this routing algorithm.

When processing the head of the query expression, as the first step, the head is decomposed into chunks. The first chunk starts at the beginning of the query expression. The consecutive chunks start at each broadcast or multicast expression. If the query does not contain any broadcast or multicast expression, then the whole query consists of only a single chunk.

As an example, remember the tree structure in Figure 4.4 on page 44. The following expression could be used to query the data from all generators with id 'k2' in a certain subdivision:

$$\underbrace{\texttt{/utility[id='a']/country[id='de']/subdiv[id='by']}}_{1}\underbrace{\texttt{/*/generator[id='k2']}}_{2}\underbrace{\texttt{//}}_{3}$$

Using related tree traversal algorithms, at least six hops would be necessary to resolve the query, starting at the `utility` node. As every path starts at the root node, the peer being responsible for that node would become a hot spot.

With the chunk based approach, the first chunk is used directly in order to address the node for `subdiv[id='by']`. That way, that node in the tree structure is addressed directly without the need for contacting the peers being responsible for the `utility` node or the `country` node.

Starting with the `subdiv` node, the query expression is recursively forwarded to the peers processing the next chunks. Along with the query expression, the offset of the current chunk is sent, indicating what part of the query has already been resolved, and what part still needs to be processed.

When a peer processes a chunk, the following cases are distinguished:

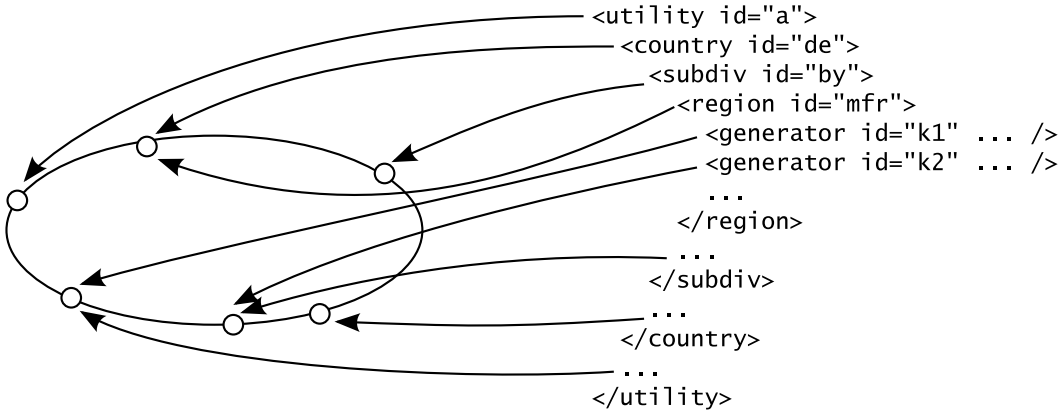


Figure 6.14.: Distributing the Tree Nodes in the DHT

1. If the chunk is a singlecast like `/utility[id='a']/country[id='de']/...`, then the addressed node can be looked up directly in the DHT.
2. If the chunk starts with a multicast as in `/*/generator[id='k2']`, then the next hops are determined as follows: For each child node, the child node's path is concatenated with the tail of the current chunk following the multicast expression. All path expressions found that way are looked up. Some of these expressions might not exist, and they are ignored. For the remainder of the expressions, the query is forwarded to the peers being responsible.
3. If the chunk is a broadcast `//`, the peer looks up all child nodes, and forwards the query to the peers holding the child nodes without incrementing the offset of the current chunk. That way, the next peers will still see `//` as the first expression of the current chunk and look up all child nodes recursively.

The algorithm above processes singlecasts in a recursive way, while broadcasts and multicasts are processed in a parallel way. Figure 6.14 shows an example of how a tree structure is distributed on a DHT, and Figure 6.15 shows how a query is resolved on that tree.

Routing of Responses. When resolving a query, it is not necessary for the involved peers to keep connections open until a response is received. Rather, the algorithm is stateless, which means that each message contains all the information needed for processing the message, which greatly fosters the reliability of the control infrastructure.

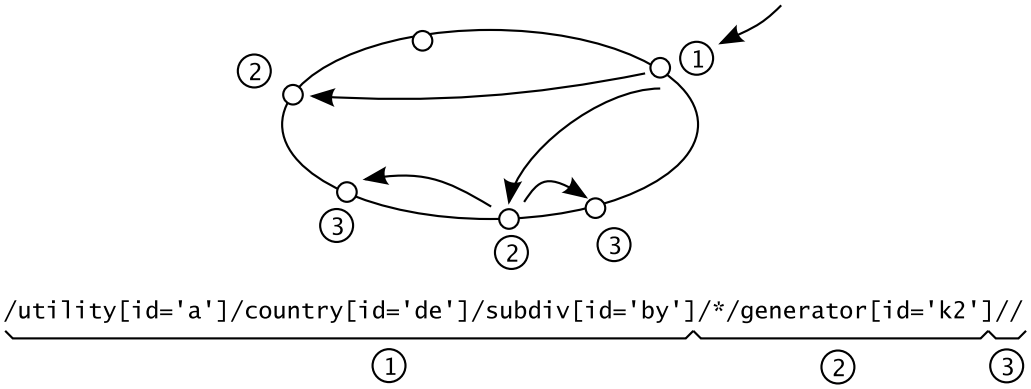


Figure 6.15.: Resolving a Query with Three Chunks

The idea behind this is to maintain a **back-route list** that is sent along with each message. Each peer adds two items to the list: First, the path expression of the current node, and second, the number of peers that were addressed as next hops in case of broadcast or multicast expressions.

Given that list, each peer receiving a response message learns the path of the next hop for forwarding the response. In addition to that, the peer learns the number of peers that were addressed in case of broadcast or multicast expressions. When a peer receives a response message, the peer waits until an equivalent number of other peers add their responses, and forwards the aggregated result to the next hop. A timeout is used to avoid that the failure of a single host can block the entire result. If a timeout occurs, a timeout flag is set indicating that the result is incomplete, and the path expression of the node is added, indicating which part of the result is missing.

The peer being responsible for the first chunk in the query is the last entry in the back-route list. That peer finally aggregates the entire relevant subtree to be returned to the utility's client.

Locality. With the back-route approach described above, any peer is able to handle any response message without any previous knowledge of the query route. This can be exploited to optimize routing distances. The idea is to use different routes for queries and responses.

The route of a query is determined by the path expressions of the relevant nodes. The peers that are responsible for a node are determined by the hash value of the node's path expression. As hash values are random, this might result in unnecessarily long routing distances.

However, the route of the response is determined by the back-route list. The

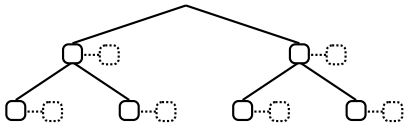


Figure 6.16.: Optimization: Using Shadow Nodes to Exploit Locality When Routing Responses

idea is to publish a **shadow node** in addition to each original node in the DHT. The key of the shadow node is deliberately chosen in a way that the physical location where the shadow node is stored is close to the region that is represented by the node in the tree structure. Figure 6.16 shows a tree with an additional shadow node for each node.

Each node must store the key of the corresponding shadow node. When a peer is resolving a query and is processing the current chunk, it would normally append the current node's path expression to the back-route list before the query is forwarded to the next hops. Now, instead of appending the current node's path, the peer adds the current node's shadow key.

That way, the response is routed back on an optimized route of shadow nodes. This is particularly useful with *schedule* queries, where a single query triggers responses that are repeated periodically. The concept of shadow nodes fits well with locality-preserving DHTs, like Pastry [91] or Tapestry [117], because with these DHTs even the replicas of the shadow nodes would be stored on peers that are close[7] to the original peer.

Summary of the Algorithm. The control infrastructure provides virtual trees that are physically distributed among peers building a DHT. The algorithm presented here can be used for resolving queries using the tree structure. The algorithm does not require the peers to keep connections open until a response is received. Rather, the response messages can be processed independently of the query messages. This enables the use of replication components complementing the routing infrastructure, which greatly enhances the robustness of the application, as evaluated in Section 7. Moreover, the approach provides enhanced efficiency using chunk-based routing and locality.

[7] The measurement of how "close" two peers are is defined by the proximity in terms of latency, as in the Vivaldi [26] algorithm.

6.4.5. Related Work

In this section, an overview of related work in peer-to-peer research is given. Moreover, other research topics related to distributed multicast infrastructures are introduced.

Distributed XML Processing on Top of Peer-to-Peer Overlays

There is a lot of related work addressing the processing of distributed XML on top of peer-to-peer overlays. Most related projects provide some means for processing XPath-like or XQuery-like expressions on the distributed XML. All these solutions share similarities with the service layer component presented here. However, the approach presented here is more reliable and more efficient with respect to the distributed power generation scenario.

The most extensive project addressing distributed XML is *AXML* [3]. The AXML project provides a distributed XML tree, where some nodes are regular *data* nodes, while other nodes are distinguished *function* nodes, representing calls to Web services. AXML supports XQuery processing on the distributed XML structure. If the query processor detects a relevant *function* node, it calls the corresponding Web service and embeds the result in the XML tree. AXML's Query evaluation algorithm is described in [2].

Unlike AXML, the aggregation tree presented here does not distinguish *function* nodes representing calls to Web services. The "stateless" approach presented here does not require the peers to keep connections open until a response is received. Therefore, the corresponding peer may be substituted between the request and the response, without having any impact on the tree structure.

Another difference between AXML and the solution presented here is that AXML was primarily developed for JXTA [43] as the underlying peer-to-peer infrastructure. As JXTA is not a DHT, the AXML query processing cannot use the efficient chunk-based routing approach introduced here. There are other research projects focusing on distributed XML processing on top of DHTs: KaDoP [4] is an implementation of AXML on top of a DHT, with a focus on semantic modeling of the distributed XML data.

XP2P [16] is a distributed XML store on top of the Chord DHT [106]. Though XPath queries in XP2P share similarities with the querying approach presented here, the solution differs in several points. The differences are mainly due to the fact that XP2P aims at providing a distributed XML database, while the motivation for the component presented here is to build a reliable control infrastructure. Firstly, XP2P does not support multicasts, which means that all multicasts must be implemented querying an entire sub-tree using broadcasts

`//`. Secondly, XP2P does not use the chunk-based routing. Apart from that, XP2P makes heavy use of the child positions in the XML tree, which makes it hard to remove nodes. The routing and aggregation tree presented above uses `id`-tags to address child nodes.

Related Research Topics

Apart from peer-to-peer based approaches, there has been a lot of other research focusing on distributed control infrastructures. For example, IP-layer multicast was designed to implement multicasting directly on the IP layer. However, this requires that the IP layer supports multicasting, which is not true for most IP networks. Apart from that, IP addresses are bound to hardware devices, and it is therefore impossible to use an addressing scheme that is independent of the underlying hardware.

Another related research topic is service migration. The peers being responsible for a node in the control infrastructure provide a service, i.e. the routing and aggregation service. If the peer fails, the service migrates from to another peer in the replication group. However, the routing infrastructure presented here differs from classical service migration, because usually the service to be migrated must keep a state, and the state must be migrated. In case of the routing infrastructure, no state needs to be transferred, and each peer is able to handle any message immediately.

Summary of Related Work

Above, a brief survey of related work was presented. The novelty of the routing and control infrastructure introduced in the present thesis is twofold: Firstly, the chunk-based routing enables more efficient queries, as paths do not need to be processed node-by-node. Moreover, the chunk-based approach relieves the root node, which decreases the risk for hot spots.

Secondly, the stateless approach introduced above decouples the routing of queries and responses. This enables an increased level of reliability, as peers may fail while a query is processed without affecting the execution of that query. Moreover, responses can be routed differently than the queries, which enables the introduction of optimizations, like locality-aware routing.

6.4.6. Summary and Outlook

In this subsection, the service layer methodology derived from the architecture in Section 3 was applied to the distributed power generation scenario in

Section 4. The gap analysis showed which of the application requirements can be fulfilled using components from Section 5, and which requirements are open issues.

The gap between state-of-the-art and the application requirements was closed with a service layer component providing a routing and aggregation infrastructure. When combining that component with the replication and PKI components from Section 5, all open requirements from the control application for distributed power generation are fulfilled.

The routing and aggregation component is designed in a way that it can be combined with replication components to provide the high level of reliability required in the application scenario. The peer-to-peer approach enables the reliability without the need for unused stand-by hardware. All peers building the routing infrastructure take over network traffic load.

A comparison with related work on distributed XML processing was presented. The major differences between related work and the component presented above are due to the fact that related work focuses on distributed XML data stores, and not so much on routing and aggregation of sensor and control data. The novelty of the component presented above is twofold: Firstly, the chunk-based approach allows more efficient query processing, and relieves the traffic load on the root node. Secondly, the stateless approach enables more reliable routing, and allows the implementation of advanced features, like locality-aware routing.

Future work includes the investigation of time constraints, in order to give some guarantees for how long it may take to process a query. Another interesting future topic is further optimization. For example, if a peer is responsible for more than one node in the tree structure, parts of the routing algorithm could be skipped.

Further research must be invested in the aggregation capabilities of the infrastructure. So far, only responses for given queries are aggregated, which means that the routing infrastructure operates in pull mode. It would be interesting to investigate how power generators could actively push data to the power company, e.g. for reporting threshold violations.

In Section 7, the routing and aggregation component will be evaluated, and the level of reliability achieved with a combination of that component and replication will be determined.

6.5. Summary

In this section, the methodology derived from the component-based architecture in Section 3 was applied to the three application scenarios from Section 4.

The contribution of this section is twofold. Firstly, three new service layer components were presented, closing the gap between state-of-the-art described in Section 5, and the application requirements presented in Section 4. The new components enable peer-to-peer-based decentralization in the three application scenarios, which has not been possible before.

Secondly, a methodology was introduced to identify the gap between the state-of-the-art and the application requirements. The idea is to decompose the service layer into generic, domain independent building blocks that can be combined when implementing peer-to-peer based applications. Relating these building blocks with the requirements yields the missing components that are to be developed. The methodology significantly reduces the development time as compared to the development of monolithic peer-to-peer applications.

The evaluation presented in the next section is divided into two parts. Firstly, the new components introduced here are evaluated. Secondly, the overall methodology is evaluated.

7. Evaluation

7.1. Service Layer Components vs. Architecture & Methodology

As defined in the Britannica's dictionary[1], evaluating something means determining its significance, worth, or condition. In this section, the two major contributions of this thesis will be evaluated:

1. The service layer components that have been developed in Section 6, enabling the application scenarios introduced in Section 4.

2. The architecture for peer-to-peer-based, decentralized applications presented in Section 3, and the methodology for developing new applications that was derived from that architecture and used throughout this thesis.

The evaluation of the service layer components is based on the application requirements defined in Section 4. These requirements represent hard evaluation criteria. For example, the evaluation determines if the maximum number of entries per peer in the user directory is below 300, which was one of the requirements.

In contrast, the evaluation of the methodology and architecture is based on soft criteria. As part of this thesis, the methodology and architecture was applied to the three industrial application scenarios introduced in Section 4. Based on the experiences made, conclusions are made about which parts of the methodology can be generalized, and which experiences are application specific.

This section is organized as follows: First, a framework is presented classifying different methods and levels of evaluation. Then, the three service layer components are evaluated. Finally, the methodology and architecture are evaluated. The summary and conclusion recapitulates the results and puts them in a broader context.

[1] http://www.britannica.com/dictionary, 2 August 2007

7.2. Evaluation Approaches

This subsection starts with an overview of different approaches for evaluating software components, and motivates the approaches used in this thesis.

The objective of evaluating a software component is to find out if the component fulfills the application requirements. If the requirements are fulfilled, the interrelationships on the service layer need to be studied to verify that the properties still hold when the component is combined with other components on the service layer. There are three approaches for evaluating software components [9]:

1. **Direct experiments** in the real world are obviously the best way to evaluate software. Whenever possible, direct experiments should be preferred over simulations [8]. Direct experiments do not rely on simplified models of the reality, and are thus immune to falsifying assumptions when building a simulation model. In distributed systems research, the Planet Lab [71] provides a platform for evaluating new technologies.

2. **Analytical evaluation** is appropriate when the evaluation can be formulated mathematically. As opposed to direct experiments, mathematical analysis is always based on a simplified model of reality, and the results may be invalid if important details of the real world have not been taken into account. However, analytical evaluation is preferred over simulation, because it yields exact results with respect to the model.

3. **Simulation** is the most common way to evaluate software components, because in most cases direct experiments are not feasible, and analytical evaluation is impossible. As the analytical evaluation, simulation relies on a simplified model of the reality. The model is implemented and run as a computer program. Multiple runs with different sets of parameters increase the statistical significance of the simulation.

In the evaluation of the service layer components in this section, all of the approaches above will be used. When possible, the components are tested in the real-world application, or at least analyzed analytically. In case this is not possible, simulations are used.

7.2.1. Levels of Abstraction

Simulation, as well as analytical evaluation, is always based on a simplified model of the real world. In this section, the abstractions implied by these models are investigated further. The goal is to make clear which assumptions

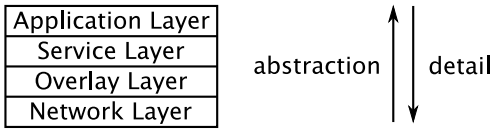


Figure 7.1.: Layers of Peer-to-Peer Applications

have been made when evaluating the service layer components presented in Section 6.

Figure 7.1 shows the layered architecture that was presented in Section 3. The level of detail being simulated when evaluating a service layer component depends on how many layers are simulated, and how many layers are abstracted.

Need for Abstraction

It is not possible to simulate all layers involved in real world applications. Trying to do this would require to take the network layer into account. However, the network layer splits further into different sub-layers. For example, the TCP/IP protocol suite consists of a transport layer (TCP, UDP), an Internet layer (IP), and a link layer (Ethernet) [17]. The link layer can be further split into a data link layer and a physical layer. Simulating the physical layer would require to simulate the electro-magnetic processes on the network interface, which is obviously not feasible. Therefore, some level of abstraction is needed.

Network Layer

Most peer-to-peer implementations being available provide some simulation features, each of which defining a different level of abstraction. The most detailed simulators are based on the NS-2 network simulator, which is a discrete event simulator targeted at networking research. NS-2 provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless networks[2].

Publications using NS-2 as the underlying network simulator usually say that the number of simulated peers per physical machine is around 100 [47, 119]. From the experience with the higher level simulations performed in this thesis it seems very ambitious to simulate 100 peers on the network layer. The number being feasible for real-world experimentation is probably even less than 100.

[2] http://www.isi.edu/nsnam/ns, 13 August 2007

Overlay Layer

There are two ways to increase the number of simulated peers. Firstly, distributed simulators can be used [47], like PDns [89], which is a parallel distributed version of NS-2. Secondly, the level of abstraction can be increased.

Message-level simulators abstract the underlying network layer, and simply assume that any peer may send messages to any other peer. The term message is used to denominate messages in the overlay protocol, not IP packets. Each message exchanged may trigger several IP packets to be sent. The main measure in message-level simulators is the number of messages exchanged for each peer, which is an estimate for the network traffic load.

Implementing message-level simulators for an existing peer-to-peer implementation is very straightforward. The only step necessary is to implement a proxy layer encapsulating the network communication. Then, the real network communication can be replaced with simulated network communication without the need for altering the rest of the application. This approach was taken in the RMF [93], which includes a **mockup transport** simulating network sockets.

However, this approach may produce simulation results that are not reproducible. Each peer is run as a separate threat or process on the host operating system. The order of the messages exchanged may depend on the scheduling of the processes by the operating system. In order to generate reproducible results, the peers must be synchronized, and all events must be scheduled and triggered by a global clock. An example for a more sophisticated message-level simulator is presented with the Simp overlay simulator in Subsection 7.2.2.

Higher Layers

Higher level simulators are used to evaluate specific properties, where it is not necessary to simulate every single message sent in the peer-to-peer protocol. For example, in order to evaluate the distribution of the data load in a peer-to-peer application it may be sufficient to model the DHT as a static hash table, and to omit the effects of churn and stabilization. Moreover, higher level simulations must be used for evaluating large-scale components, where the simulation of every single message would be too costly.

Summary of the Abstraction Levels

Above, an overview was given of the different abstraction levels for simulations. As simulation is the most common way for evaluating software components, the Simp is presented below as an example of a message-level overlay simulator. The Simp was developed at Siemens Corporate Technology as part of this thesis.

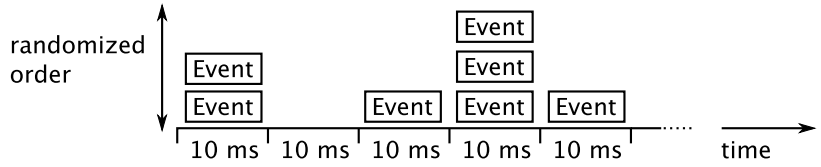


Figure 7.2.: Event Queue for a Peer in Simp

7.2.2. Message Level Simulation

In this section, the **Simp** project is introduced, contributing an example of a message-level simulator. Most of the concepts implemented in Simp can also be found in related work on simulation, which makes Simp a minor contribution to this thesis. However, the Simp provides a good example of an overlay simulator. The goal of this section is to give an insight into event-based simulation.

The Simp project was started as part of this thesis, because most related simulators are very large and complex. For example, the **p2psim**[3] simulator used at the MIT consists of more than 40 000 lines of C++ code. The Simp project aims at a simple and lightweight simulator.

Architecture

Simp is an event-based simulator providing reproducible results. Two design decisions were made to provide reproducibility: Firstly, all random number generators used in Simp are configured in a configuration file. If the same seed is used in two different runs, the random number generators produce the same sequence of random numbers. Secondly, Simp is implemented as a single-threaded application. That way, the scheduling of the operating system has no influence on the results of the simulation runs.

The basic data structure in Simp is the event queue, as illustrated in Figure 7.2. The event queue is divided into time frames. The duration of a time frame is a configuration parameter, and is typically set to 10ms[4].

There is one event queue for each simulated peer[5], and one global event queue managing the arrival of new peers. The main operation of the simulator is to inspect the current time frame in all event queues, and to trigger the

[3] http://pdos.csail.mit.edu/p2psim, 13 September 2007

[4] The reason for choosing 10ms is that it takes at least 10ms for a message to be sent over a simulated network link, which makes it unnecessary to simulate shorter time frames.

[5] In the plain version of Simp, a single event queue shared by all peers would be more efficient. However, having one queue per peer enables the implementation of a distributed version of Simp, as described below.

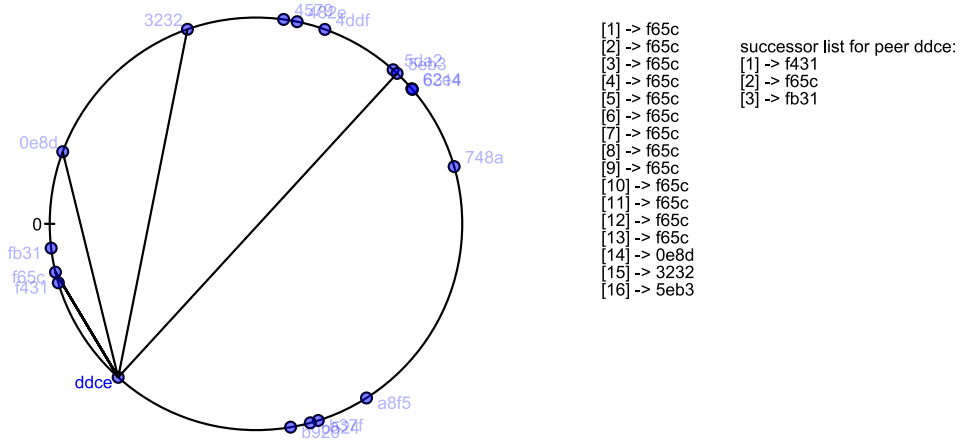


Figure 7.3.: Example of Snapshot Dumped by Simp's Logfile Evaluation

events' call-back methods. If there is more than one event in a time frame, the events are processed in random order.

An evaluation and comparison of different event queue algorithms was conducted as a cooperation project with Siemens Corporate Technology and the University of Würzburg. The results can be found in [50].

Periodic events are scheduled by event generators. For example, there is a generator producing join events whenever a new peer arrives in the overlay. The event generators can be configured with probability distributions determining the time interval between two consecutive events. For example, the join event generator is configured with a Gaussian distribution, while the peers' lifetime is configured with a Weibull distribution.

Simp's core framework provides generic events, like events triggered when a message is received, or events triggered when a time-out occurs. The overlay-specific implementations are derived from that generic core framework. For example, an implementation of the Chord DHT derives events for all interactions defined in the Chord protocol [106], e.g. fix-fingers, stabilize, etc.

The evaluation of a simulation run is not done at runtime. Rather, at runtime all information about the overlay is dumped into a logfile. This logfile is analyzed by a stand-alone application, called the *logfile evaluator*. The logfile evaluator is a generic framework. One can derive from it when implementing specific evaluators, answering specific questions.

An example for an evaluator application is a tool that dumps snapshots of the Chord overlay into an interactive SVG file. The SVG file can be viewed to

verify the finger tables. An example of such an SVG file is shown in Figure 7.3.

Current Status and Future Work

In order to increase the scalability of the Simp, a distributed version was implemented in the Diploma thesis by Peter Hessheimer [49]. This version distributes the simulated peers among several physical hosts. The synchronization is done after each simulated time frame. An intelligent look-ahead algorithm allows the computers to continue simulation while they are waiting for a synchronization signal. The look-ahead simulation needs to be discarded only if events are received that are relevant for a time frame that has already been simulated.

However, with respect to the evaluation of the service layer components introduced in this thesis, Simp proved less helpful. The range query component in the distributed user directory scenario must support at least 500 000 peers, which is too much to be simulated on the overlay layer, even with the distributed version of Simp. The other components can be evaluated using other approaches, e.g. direct experiments and analytical evaluation. As said above, these approaches should be preferred over simulations.

The presentation of the Simp overlay simulator above served as an example for showing the concepts used in message-level overlay simulation. However, Simp provides only a minor contribution to this thesis.

7.2.3. Summary

This subsection provided the background for the evaluation of software components. Three approaches for evaluating software were differentiated: Direct experiments, analytical evaluation, and simulation. It was found that direct experiments should be used whenever possible, and that analytical evaluation should be preferred over simulation.

However, in peer-to-peer related research, direct experiments are often not feasible, and analytical evaluation is impossible. Simulation is the most common evaluation method in this area. Therefore, simulation was introduced in greater detail. The different levels of abstraction were introduced, and the Simp was presented as an example of a message-level simulator.

In the next section, the evaluation of the service layer components from Section 6 will be presented.

7.3. Evaluation of the Service Layer Components

7.3.1. Confidential Communication in Business Collaboration

This subsection targets the evaluation of the confidential communication component, and its integration into the collaborative product design (CPD) scenario in the automotive industry. The evaluation has been performed in two steps:

1. A pilot implementation of the CPD scenario has been contributed to the ATHENA project as part of this thesis. The pilot provides a real-world integration of the CPD scenario with FIAT as the OEM, and Siemens as a first tier supplier. This enabled direct experiments in a real-world implementation.
2. The confidential communication component has been prototypically integrated into the peer-to-peer platform underlying the pilot project. That way, the feasibility of the onion routing approach is shown, and the impact on performance and security is evaluated.

In the following, first the pilot implementation and the evaluation of the CPD scenario is described, then the integration of the service layer component is evaluated.

Evaluation of the CPD Scenario

As part of this thesis, a pilot implementation of the CPD scenario was contributed to the ATHENA project [81]. The pilot was set up jointly by FIAT and Siemens Corporate Technology, and integrates work that was done previously in the ATHENA project. The evaluation of the pilot shows the feasibility of the peer-to-peer-based approach [81], and yields the requirements to be fulfilled when integrating the confidential communication component. In the following, the pilot implementation is introduced and evaluated.

OEM Side

Figure 7.5 shows the architecture of the pilot implementation on the OEM side. The XSRM on the left is a Web-based application used by FIAT for creating RfQ documents and for processing quotations. The XSRM uses several databases, but these are not visible to the supplier. By integrating the XSRM, FIAT ensured that the pilot implementation is provided with real-world RfQ data.

The generated RfQ documents are processed by a business process engine, which is composed of the tools Maestro, Nehemiah, Johnson, and Gabriel.

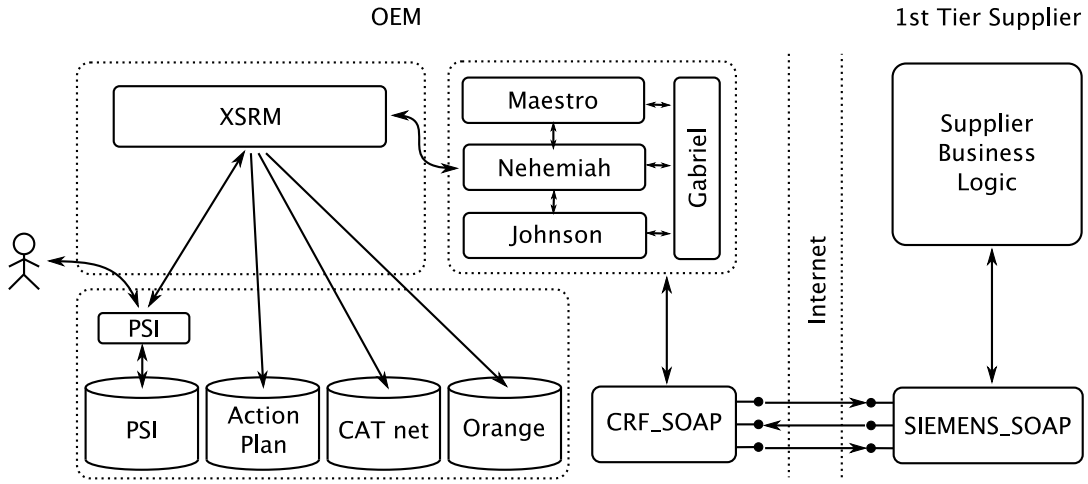


Figure 7.4.: Architecture of the Pilot Implementation: OEM Side [81]

These tools are provided by SAP as part of their contribution to the ATHENA project. The tools enable platform independent modeling of collaborative business processes. The CPD process was modeled jointly by FIAT and Siemens. The process execution on the OEM side is implemented using a client-server architecture, and the execution on the supplier side is implemented on top of the Business Resource Management Framework (BRMF) [42], which is a decentralized peer-to-peer-based business collaboration platform.

The following is a short survey of the tools and the concepts used to model the collaborative business process in the CPD scenario, and to execute it:

Maestro [80, 78] is a tool for modeling executable business processes in a platform independent way. When modeling collaborative business processes, Maestro allows to distinguish a public view and private views. The public view defines the steps where the business partners need to interact. For example, the CPD process contains a step where the Quote is handed over from Siemens to FIAT. The private views define the internal processes of each business partner, and they do not need to be visible to the other partners.

Nehemiah [75, 74] provides an enactment engine for collaborative business processes as well as simulation functionality. Nehemiah is used to execute the processes modeled in Maestro. While Maestro is used at design time, Nehemiah and the other tools form the runtime environment.

Johnson [73] implements the actual messaging when the business partners

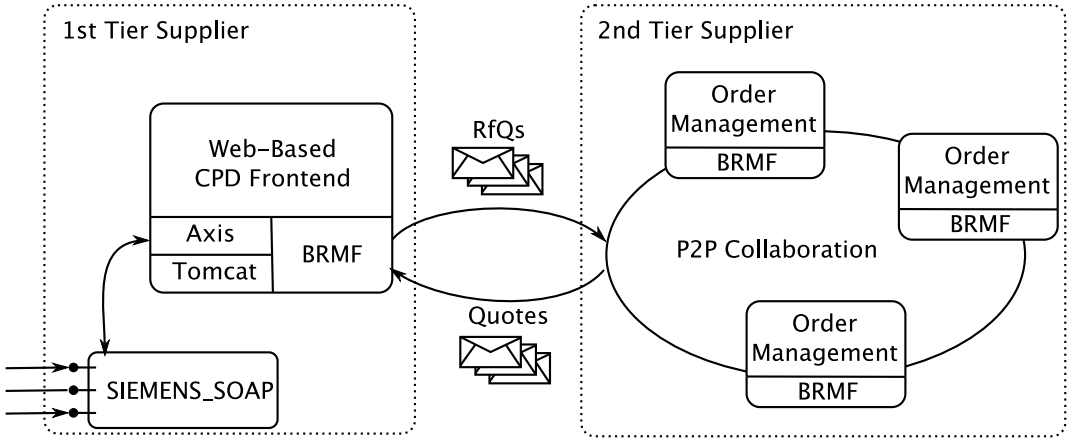


Figure 7.5.: Architecture of the Pilot Implementation: Supplier Side

interact. In the automotive pilot implementation, Johnson was configured to invoice Web services in order to retrieve the business partners' data.

Gabriel [75, 77] provides an abstraction layer for the data used within the process. In the automotive pilot, Gabriel is configured with the WSDL files from the business partner's web services. At design time, Gabriel is accessed from Maestro when the process relevant data is defined. At runtime, Johnson hands over this data to Gabriel. Nehemiah uses the data provided by Gabriel when conditional steps are performed in the business process.

The implementation described above allows the execution of real-world CPD processes using real-world RfQ data from the automotive industry. This provides the framework for evaluating the requirements for confidential communications in a real-world pilot application. The supplier side integrates a peer-to-peer-based business collaboration platform, as described below.

Supplier Side

Figure 7.5 shows the architecture of the pilot implementation on the supplier side. The collaboration of the first and second tier suppliers is based on the **Business Resource Management Framework (BRMF)** [42], which is a peer-to-peer based business collaboration platform resulting from an earlier subproject of ATHENA [79, 76].

The BRMF is based on the **Resource Management Framework (RMF)** [93], which is a DHT implementation developed by Siemens. As the RMF is applied in a real-world piloting project, it serves as the framework for evaluating the confidential communication component.

The first tier supplier forms the bridge between the process-driven Web service environment on the OEM side, and the event-driven peer-to-peer environment on the supplier side. Therefore, the implementation of the first tier supplier has two communication stacks in parallel: The Web service stack is based on **Axis**[6] and **Tomcat**[7], and the peer-to-peer stack is formed by the BRMF library.

The main business task of the first tier supplier is to split the OEM's RfQ into different sub-RfQs to be published in the peer-to-peer collaboration platform. The quotes received from the second tier suppliers need to be aggregated, and a final quotation needs to be created and to be sent back to the OEM. The conversion of the data between the Web service representation and the BRMF representation is done using XML as the intermediate format, and XML Schema as the format definition. This is straightforward, because Axis as well as the BRMF use XML and XML Schema for the internal representation of their data. The BRMF implements an extended version of the **Castor**[8] library, which generates Java-to-XML bindings based on XML Schema definitions.

A Web-based CPD front-end can be used to view RfQs from the OEM, and to issue RfQs for the second tier suppliers. When quotes from the second tier suppliers are received, the Web front-end can be used to compose a response for the OEM.

As there are no second tier suppliers participating in the ATHENA project, the second tier suppliers are simulated using applications based on the BRMF. A simple graphical order management interface allows the second tier suppliers to subscribe and receive RfQs, to issue change request, and to generate quotations.

Results of the Pilot Evaluation

Above, the pilot implementation was described. The goal of evaluating the pilot is to proof the validity of the use case, and to get a framework for the confidential communication component in a real-world scenario. The integration of confidential communication in the RMF will be used for evaluating the onion routing approach based on the real-world framework.

The pilot itself is evaluated by running different test scenarios. The scenarios

[6] `http://ws.apache.org/axis`, 2 December 2007
[7] `http://tomcat.apache.org`, 2 December 2007
[8] `http://www.castor.org`, 12 September 2007

and the results are described in the corresponding ATHENA Deliverable [81], and will not be detailed here. The following is a brief overview of the main outcomes:

- Distributed search and subscription handling works successfully, and requires low bandwidth. The average search time is well below the ATHENA limit of 10 seconds.
- Collaborative RfQ processing performs well, the workflow was successfully implemented. It was shown that a peer-to-peer-based business collaboration platform can be seamlessly integrated with the traditional process-driven collaboration on the OEM side [103].
- Dynamic handling of update events works as expected, the time for subscriptions to be triggered is well below the ATHENA limits.

Especially the low bandwidth and data load observed in the evaluation are a good basis for the implementation of the confidential communication component. In the remainder of this section, the implementation of confidential communication in the RMF is described, and the impact on the performance as well as the level of security are evaluated.

Evaluation of the Confidential Communication Component in the RMF

As shown above, the RMF was the basis for a piloting implementation of the CPD scenario that was contributed to the ATHENA project as part of this thesis. The implementation of confidential communication within the RMF is therefore a valid way for evaluating confidential communication as a service layer component in CPD.

Figure 7.6 on the left illustrates RMF's message encoding[9]. The messages are encoded in a proprietary binary format implementing a type-length-value approach, similar to the Basic Encoding Rules (BER) for the SNMP [110]. Onion routing is implemented by introducing an additional type field, indicating encrypted onion routing data.

On the right side of Figure 7.6, the building blocks of the onion routing implementation are shown:

1. The **interception handler** is used to intercept locally generated messages and wrap them into the onion layers. This is implemented in RMF's remoting layer, where low level message handling is performed.

[9] RMF actually adds more headers to the messages, but the additional headers are omitted in Figure 7.6.

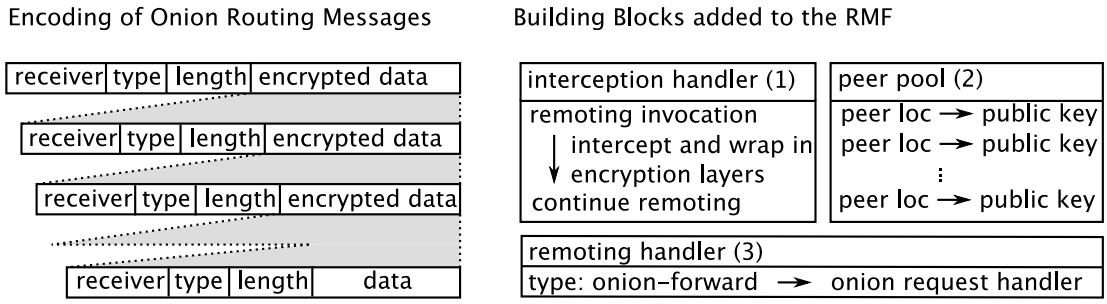


Figure 7.6.: Onion Routing Implementation in the RMF

2. The **peer pool** stores random peer locators and the corresponding public keys. The pool is used for establishing the random routes used for onion routing. It must be continuously updated in order to discard entries for peers that are no longer reachable.

   In the RMF, peer locators store IP addresses, peer identifiers, and some additional information. There are two ways for maintaining that pool: Firstly, the pool can be supplied with identifiers and keys arriving incidentally, e.g. with the stabilization protocol or with the finger table updates. Secondly, the peer can actively query random identifiers and public keys periodically. While the second approach is cryptographically more secure, the first approach does not cause additional network traffic. As the experiences from the pilot implementation show that network traffic is not a major bottleneck, the second approach is recommended.

3. The **remoting handler** is called when encrypted messages are received. The handler decrypts the messages, and forwards them to the next destination. RMF provides a generic request handler framework where the new handler can be added easily.

The underlying VPN was omitted in the evaluation, and it was replaced by a simple static PKI infrastructure. In the evaluation, each peer was simply configured with the public keys and the peer locators of all other peers.

Performance

Onion routing relies on routing documents along a random path instead of publishing them directly. As described in Section 6, a path length of 5 is appropriate in the CPD scenario. On the overlay layer, each lookup operation

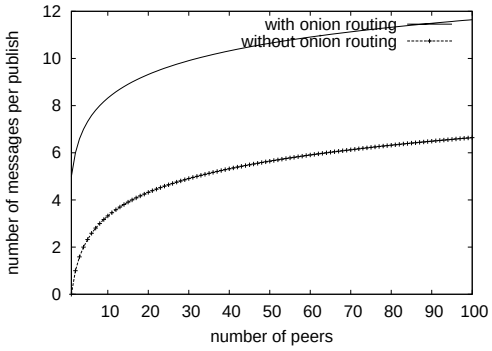


Figure 7.7.: Number of Peers vs. Number of Hops Per Lookup for $c = 5$

takes several hops, i.e. several peers need to be connected to perform a lookup operation. As shown in Appendix A, the number of hops for each lookup is $\log_b n$, where n is the number of peers, and b depends on the design of the overlay's routing tables. Let c be the length of the onion routing path. When publishing a document, the onion routing component increases the number of hops by c as compared to the raw overlay protocol. Figure 7.7 shows the number of hops with and without onion routing for $b = 2$ and $c = 5$.

The number of collaborating business partners n is expected to be in the range from 25 to 100. As shown in Figure 7.7, adding onion routing roughly doubles the latency in this range. For larger n, the impact of onion routing decreases. The impact factor on latency converges slowly to one:

$$\lim_{n \to \infty} \frac{\log_b n + c}{\log_b n} = 1$$

The evaluation results of the piloting project in ATHENA shows that doubled latencies are still well below the ATHENA limits.

Additional network traffic is added by continuously updating the pool of peer locators needed to build the onion routes. In order to keep this pool up to date, the frequency of the updates must be related to the mean online time of the peers. As the churn of business partners can be expected to be comparatively low, the additional traffic can be neglected.

Reliability

When publishing resources directly, the sender gets direct feedback if the document reached the destination successfully. Using onion routing, the resource is sent to the first hop and there is no way for the sender to learn whether a resource finally reaches its destination successfully, or if a node on the path fails forwarding the resource. Implementing acknowledgments via circuits solves this problem, as discussed in Section 6. If circuits are not implemented, the onion routing can only implement a best effort service. However, with respect to the CPD scenario it is feasible to apply a best effort approach, and to implement application-level acknowledgments, sent from the OEM directly to the suppliers and vice versa.

Confidentiality

The level of confidentiality achieved can hardly be determined by measurements. Using a path length of 5 hops, at least 5 business partners need to conspire in order to break the confidentiality. Therefore the threat scenarios described in Section 6 become very unlikely when using the confidential communication component. However, security is a moving target. Further threat scenarios will arise in the CPD application, and new components will need to be developed defeating these threats.

The contribution of the confidential communication component is to show that security problems in peer-to-peer based business collaboration can be solved. This makes peer-to-peer approachable for electronic collaboration and marketplace scenarios. By means of this component, one step was taken towards enabling businesses to benefit from decentralized management and coordination, self-organization and resilience of peer-to-peer systems without loosing the ability to communicate confidentially. Other steps will become necessary when other security threats come up.

Summary, Conclusions and Outlook

This subsection presented the evaluation of the confidential communication component introduced in Section 6. The use case scenario was implemented as a pilot project, set up jointly by FIAT and Siemens. The pilot implementation was based on the RMF, which is a peer-to-peer platform developed by Siemens. The implementation allowed to perform real-world experiments and to derive the requirements for the new service layer component.

The confidential communication component was then implemented in the RMF, and it was shown that the latency and traffic load stays below the limits

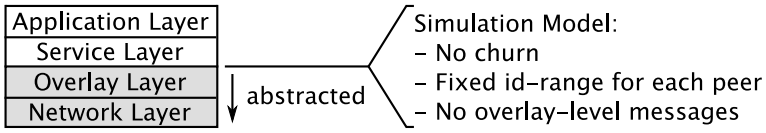


Figure 7.8.: Simulation Model for the Evaluation of the EPHT Component

required by the pilot specification. It was shown that the threat scenarios described in Section 6 become very unlikely when the component is integrated.

In the future, more threat scenarios will be identified, and new components will need to be integrated to keep the application secure. The work presented here is a starting point for future security components, and the component-based architecture will make it easier for future developers to estimate the impact of their components on an existing application.

7.3.2. Search Index for a Distributed User Directory

The evaluation of the confidential communication component above was mostly based on direct experiments, which was made possible through the piloting activities in the ATHENA project. In this subsection, the service layer component providing range queries in the distributed user directory scenario is evaluated. This component is designed to be used in large-scale deployments with 500 000 users and more[10]. Therefore, higher level simulations must be used in the evaluation.

The evaluation data is taken from a German phone book CDROM from 1997 [109], because newer electronic phone books restrict data export due to privacy regulations. For the evaluation presented here, the entries for the city of Munich are used, which has 620 853 entries. As each peer is supposed to provide only its own entry, the number of peers is equal to the number of entries.

Due to the high number of peers to be simulated, the simulation model abstracts the details of the underlying overlay. The DHT is assumed to be always stable, and churn is not modeled. Each peer has a static range of identifiers being mapped to that peer. The simulation model is illustrated in Figure 7.8.

[10] As said in Section 6, 500 000 is the requirement of Siemens Communications for the prototype of the Peer Things project [15].

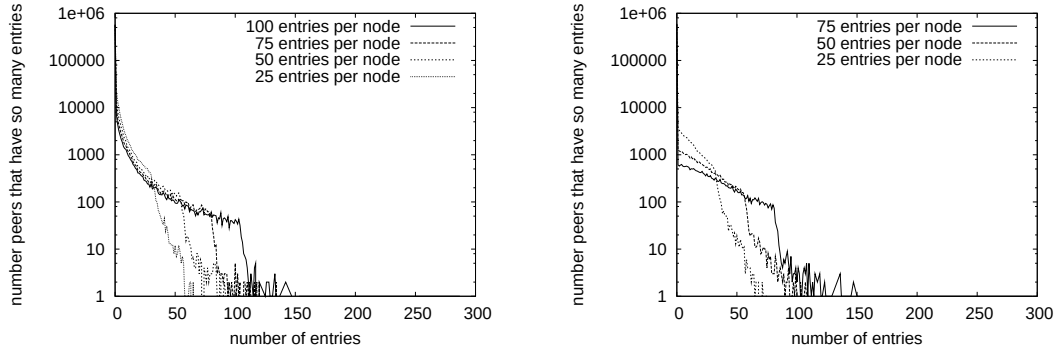


Figure 7.9.: Entries per Peer, using $n = 26$ on the left and $n = 5$ on the right [101]

Parameters for the EPHT Algorithm

As described in Section 6, the EPHT algorithm has two parameter determining its behavior:

1. m is the maximum number of entries that can be stored in the root node. Increasing m means to store more data per node, and decreasing m means to distributed the tree on more different peers.

2. n is the number of children per node. Increasing n results in an increased width of the tree, and decreasing n results in an increased depth of the tree.

As part of this evaluation, the optimum values for m and n with respect to the evaluation data are determined.

Apart from these design-time parameters, the initial prefix length was introduced as a run-time parameter. The initial prefix length determines the prefix length to start with when running a new search. For example, if the initial prefix length is 5, and the user searches for SCHMIDTPETER*, then the search will start with SCHMI*, and then increase the prefix depending on the result. The initial prefix length is a run-time parameter, because it can be adapted while the tree grows.

Evaluation of the Data Load

The number of entries per peer is one of the key performance indicators, as well-balanced data are the prerequisite for good balancing of the network load.

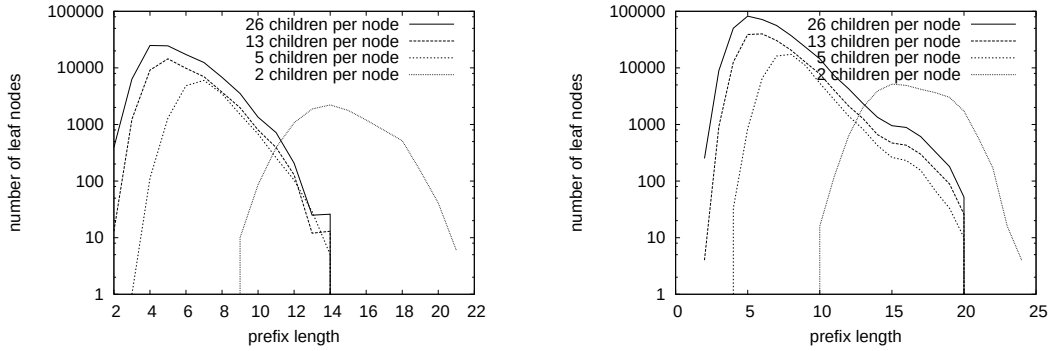


Figure 7.10.: Prefix Lengths with $m = 100$ (left), and $m = 25$ (right)

In the Peer Things project [92], a limit of 300 entries per peer was defined as the requirement being feasible for using CPE units as peers in the VoIP scenario.

As described above, the EPHT algorithm has a parameter m, determining the maximum number of entries that can be stored in the root node. Figure 7.9 shows the number of entries per peer for m in 25, 50, 75, and 100, without replication.

Note that the y-scale showing the number of peers is logarithmic. Nearly all of the 620 853 peers store less than 3 entries. No peer stores more than 150 entries. Assuming an average size of an entry of 128 Bytes, a peer holding 150 entries would store less then 19 kBytes. This fulfills the requirements of the Peer Things project. It is feasible to handle 19 kBytes even on embedded devices with a built-in peer-to-peer stack, and it is easily possible to replicate 19 kBytes through current Internet connections.

Evaluation of the Initial Prefix Length

In Section 6, a value of 5 was proposed for the initial prefix length. Figure 7.10 shows why 5 is a good value. 5 is the average prefix length for n in 5, 13, and 26. Only binary EPHTs with $n = 2$ result in a significantly larger average prefix length. If the average prefix length changes over time, e.g. if the number of users or the distribution of names is other than expected, then the initial prefix length can be adapted at runtime without the need for re-shaping the tree structure.

	SCHN*			OLPP*		
	n = 5	n = 13	n = 26	n = 5	n = 13	n = 26
m = 50	1068	958	495	99	3	4
m = 100	590	670	279	39	3	4

Table 7.1.: Number of Lookup Operations

Evaluation of the Network Traffic

The network traffic is evaluated in terms of the number of lookup operations in the DHT that are needed to process a range query. Of course, the number of lookup operations depends on the number of entries matching the queried prefix. For example, querying the prefix SCHN* results in 4683 entries, and OLPP* yields only a single entry.

However, querying SCHN* is an artificial example, as real-world applications would probably abort that query after a certain number of results is retrieved, and ask the user to formulate the query more specifically. Therefore, the query for OLPP* is a more realistic example. Table 7.1 shows the number of lookup operations without caching. With respect to the parameters m and n, the table above can be interpreted as follows:

m: Increasing the maximum load of the root node m results in less nodes to be looked up. This is evident, because with higher values for m the data is distributed among a lower number of different nodes.

n: Increasing the number of children per node n results in a lower number of lookup operations. To exemplify this, imagine a search for OLPP*. In a tree with n = 5, the application searches all entries matching the prefix [K-O][K-O][P-T][P-T]. People with a last name starting with *Lost* would match the same prefix as *Olpp*. The result *Lost* would be filtered out on the application layer, but not by the service layer component. On the other hand, with n = 26 the search would be [O-O][L-L][P-P][P-P]. In that case, *Lost* does not match that prefix. Therefore, a higher number n results in less false matches.

The significance of this effect becomes clear for low numbers of n, like n = 5. The number of entries matching [K-O][K-O][P-T][P-T] in the phone book is 1801, which explains the overhead of 99 lookups in Table 7.1. For n = 26, there is only one matching entry, and this entry is found using only 4 lookups.

These results suggest that the number of children per node n should be as large as possible to reduce the number of lookup operations.

	n = 5	n = 13	n = 26
m = 50	6% of 34 821	37% of 94 297	60% of 193 801
m = 100	2% of 18 353	29% of 48 757	53% of 98 501

Table 7.2.: Percentage of Empty Nodes

Evaluation of Empty Nodes

The results above suggest to choose the number of children per node n as large as possible. However, increasing n yields a larger number of empty nodes, because there are many prefixes that do not exist. Therefore, this effect must be considered before giving a recommendation for a good value of n.

The percentage of empty nodes is shown in table 7.2: As expected, the number of empty nodes raises with the number of children per node. However, even with n = 26 there are only 60% empty nodes, which is a good trade-off given the great reduction of traffic overhead for n = 26. Therefore, n = 26 can be recommended as a reasonable value.

Summary, Conclusions and Outlook

In this subsection, the evaluation of a service layer component enabling range queries in a distributed user directory scenario was presented. The service layer component is based on the EPHT algorithm, which was presented in Section 6.

The EPHT algorithm has two configuration parameters. In the evaluation, the best values for these parameters with respect to the evaluation data were determined. It was shown that using these parameters, the requirements of the Peer Things project are met.

Although the evaluation above is heavily based on data from a distributed user directory scenario, the EPHT algorithm can also be used in other scenarios where range queries on top of DHTs are required. It is up to future work to explore new fields of application. The evaluation above can be a basis for evaluating EPHTs in new scenarios.

7.3.3. Routing and Aggregation Infrastructure for Distributed Power Generation

The third application scenario presented in Section 4 is a reliable control infrastructure for distributed power generation. The gap analysis for that scenario in Section 6 revealed the need for a service layer component providing routing and aggregation functionality in a reliable way. The implementation of

that component was presented in Section 6. The following subsection presents the evaluation of this component.

There are three main motivations for choosing peer-to-peer-based decentralization as the basis for the routing infrastructure: Scalability, self-organization and reliability. The focus of the evaluation will be on reliability:

- The scalability of tree-shaped data structures was already targeted in the evaluation of the EPHT component above. EPHTs and the multicasting and aggregation tree share many similarities. For example, both algorithms use the fact that one can jump directly to some inner node of the tree, which relieves the load of the root node. In order to avoid repetition, scalability will not be the focus of the evaluation of the routing infrastructure.
- Self-organization is provided on the overlay level, and does not need explicit service layer support in the control infrastructure. However, it is important to keep the self-organization property intact, and not to dismiss it on the service layer.
- Reliability is a very interesting requirement. Firstly, the reliability of tree structures on top of a peer-to-peer overlay has not been evaluated before. Secondly, reliability is achieved with a combination of service layer components: The routing and aggregation component, and a replication component. It is therefore a good example for studying the interrelationships of two components on the service layer.

In the beginning of this section, three approaches for evaluating software components were identified: Direct experiments, analytical evaluation, and simulation. As distributed power generation is a future scenario, direct experiments are not possible. However, it is possible to evaluate the level of reliability analytically. The model underlying that analysis is described below. The analytical results are verified with simulation.

In the remainder of this section, first the effect of regular churn on reliability is evaluated, then the effect of massive node failure, e.g. blackouts, is estimated.

Evaluation of the Reliability with Regular Churn

As motivated in Section 4, peers and Internet connections are considered to be potentially unreliable, i.e. peers may fail, or the connection to the Internet may become temporarily unavailable. The goal of combining the routing component and replication is to achieve a high level of reliability in spite of this churn. The aim is to surpass the reliability of classical client-server based implementations while using inexpensive hardware.

Peer-to-peer overlays have originally been developed to support file sharing communities in the Internet. In typical file sharing scenarios, a fraction of the peers joins and leaves the overlay very quickly, which results in comparably high churn rates in these systems [107]. As DHT overlays are designed to cope with these high churn rates, the DHT-based routing infrastructure can be expected to be very reliable, as all peers in this scenario are operated by the power company, and are expected to run in a quite stable way.

Regarding the node failures, the focus in this evaluation is on **clean node failure**s, i.e. nodes become unavailable or connections become unusable. This does not include malfunctioning nodes sending invalid messages that could cause invalid routing table entries in the peer-to-peer overlay. Dealing with incorrect messages is very much related to peer-to-peer security, which is targeted by other service layer components. Related work on that can be found in [98].

The nodes of the tree structure are published as data resources in a DHT. There is a unique peer currently being responsible for each resource. Using replication, there are r peers holding replicas of the resource. The responsible peer plus the peers holding replicas form the replication group for that resource. The number of replicas r varies with the implementation; it is typically a value between 2 (as in [93]) and 8 (as in [87]).

The failure of an entire replication group results in data loss, i.e. the corresponding node in the tree structure becomes unavailable, and the tree needs to be set up again by the power company. The reliability of the tree structure can be expressed with the probability of data loss. In contrast to previous work on DHTs, the reliability is not evaluated using empirical data, but by analytical methods, deriving a formula for the probability of data loss. The formula shows the difference between the reliability of classical client-server based systems and peer-to-peer based architectures.

General Formula for Client-Server and Peer-to-Peer. As a first step, a formula is presented that can be applied in both client-server based scenarios and peer-to-peer architectures. Given a number of n peers (or servers), each of which having an average availability of $a = 99.999\ldots\%$. Let r be the number of replicas of a resource in the peer-to-peer network (or the number of backup servers in a client-server based scenario). The probability that an entire replication group fails is

$$p = (1 - a^n) \cdot (1 - a)^r$$

The first factor $(1 - a^n)$ is the probability that a peer fails, the second factor $(1 - a)^r$ is the probability that this peer's replication group fails.

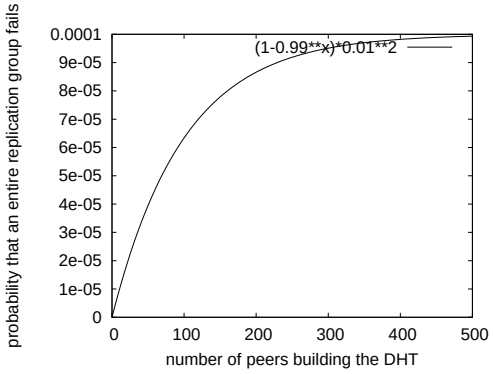


Figure 7.11.: Probability of an Entire Replication Group to Fail for $a = 99\%$ and $r = 2$.

The probability has an upper bound of $(1 - a)^r$, and the upper bound is converged with an increasing number of peers n. Increasing the number of replicas r decreases the probability of an entire replication group being failed. Figure 7.11 illustrates the probability for $r = 2$ and $a = 99\%$.

Interpretation with Respect to Peer-to-Peer. In a client-server-based scenario, the probability of data loss is usually reduced by increasing the reliability of the server. There are servers with significantly higher availability rates than 99%. Peer-to-peer-based solutions are designed for running on cheap, unreliable hardware. These solutions improve the reliability using replication components, which implement **self-healing** capabilities. In the following, the impact of self-healing on the formula above is shown.

All DHTs provide some stabilization protocol including ping messages being continuously exchanged among neighboring peers. A typical time interval between these messages is 10 seconds, as in the Bamboo implementation [87]. If more than i consecutive ping messages remain unresponded, the corresponding peer is considered to be offline. Another peer is taken into the replication group, and resources are copied to that peer. A typical value for the threshold i is 3.

In total, as a very pessimistic assumption, it takes less than one minute to detect the node failure and to transfer the replicated resources to another peer. After one minute, the replication group is established again[11]. If necessary,

[11]Note that this does not mean that the resource is unreachable for one minute. Switching

the recovery time can be decreased by increasing the frequency of the ping messages.

The low recovery time dramatically reduces the probability of data loss in peer-to-peer based systems. In a classical client-server based scenario, a server that fails must be repaired by a service engineer of the service provider hosting that server. Depending on the service level agreement, the provider agrees on how long it takes until the service engineer reacts. This results in a long time interval during which one server is missing.

Let t_p be the recovery time in a decentralized peer-to-peer environment, and t_s be the reaction time of a service engineer in a client-server-based application. The following formula shows how the probability of data loss decreases when t_p is significantly lower than t_s:

$$p = (1 - a^n) \cdot (1 - a)^r \cdot \left(\frac{t_p}{t_s}\right)^r$$

This explains why peer-to-peer systems are able to provide a superior level of reliability without using highly available hardware.

Evaluation of the Reliability with Massive Node Failure

The analysis above targeted the probability of data loss with regular churn, i.e. scenarios where the failure of two different peers is independent of each other. However, there are scenarios where several peers fail at the same time, e.g. when a blackout occurs, and the peers are all affected by the blackout.

Thinking of massive node failure, the following question needs to be answered: If k out of n peers fail at the same time, how much is the probability of data loss? This question has not been answered yet in related work about peer-to-peer systems.

The probability $p(n, k)$ is $g(n, k, r)/m(n, k)$, where $g(n, k, r)$ is the number of ways to choose k failing peers without data loss, and $m(n, k) = \binom{n}{k}$ is the number of ways to choose k peers out of n. In order to construct a recursive formula for $g(n, k, r)$, all peers can be considered in a line. The black peers are the peers that fail, the white peers are the peers that work. If the size of the replication group is $r = 3$, the constraint is that there must not be more than two neighboring black peers. An example of a still-working configuration is given in Figure 7.12.

Distributing the failed (black) peers starts at the left. For $r = 3$, one has three possibilities. First, ○····· the first peer is not failed. In that case, $g(n - 1, k, r)$ possibilities are left. Second, ●—○····· the first peer does fail, but the next one

responsibility from one peer to another peer in the replication group is significantly faster than taking a new peer into the replication group.

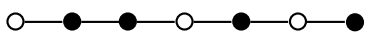
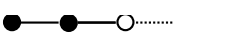


Figure 7.12.: Still-working Configuration of Failed (black) and Working (white) Peers

does not. In that case $g(n-2, k-1, r)$ possibilities are left. Third,  the first two peers fail, and the third one does not. In that case $g(n-3, k-2, r)$ possibilities are left. For $r = 3$, these are all possibilities for the first step.

This consideration leads to the following recursive formula[12]:

$$g(n, k, r) - c(r) = \begin{cases} 0 & \text{if} \quad n < 0 \\ 1 & \text{if} \quad \begin{array}{l} n = k \text{ and} \\ 0 \le n \le 2 \end{array} \\ \sum_{i=1}^{r} g(n-i, k-i-1) & \text{otherwise} \end{cases}$$

It is clear that $g(n, k, r)$ is well-defined for all $n, k, r \in \mathbb{Z}$ with $r > 1$, because n decreases with each step of the recursion, and the recursion terminates when $n < 0$. The formula yields $g(n, k, r) = 0$ for $n < k$, which means that there is no way to distribute a number of failing peers k that is greater than the overall number of peers n.

Figure 7.13 shows the probability of data loss depending on the percentage of the peers that fail at the same time. The x-axis is cut at 66%, because if more than $2/3$ of the peers fail the probability for data loss is always 100%.

The correction $c(r)$ must be substracted, because the actual DHT is a ring structure, and not a line. Therefore, $g(n, k, r)$ counts some cases that would lead to data loss in a ring structure. For $r = 3$ there are three such cases: Firstly, one peer fails at the beginning of the line and two peers fail at the end of the line. Secondly, two peers fail at the beginning of the line and one peer fails at the end of the line. Thirdly, two peers fail at the beginning of the line and two peers fail at the end of the line. These three cases would lead to data loss if we had a closed ring.

In general, the correction $c(r)$ is

$$c(r) = \frac{1}{2} r(r-1)$$

However, this correction becomes insignificant for large n.

[12] The correction $c(r)$ is explained below.

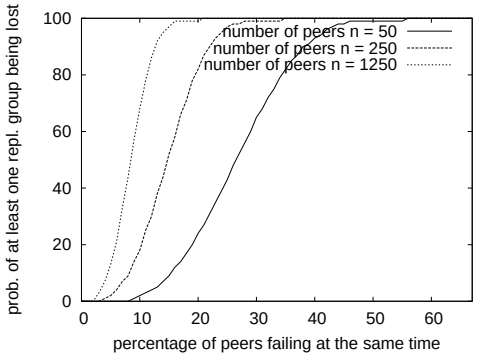


Figure 7.13.: k of n Peers Failing at the Same Time

Effect of Locality on the Formula Above. The consideration above is based on the assumption that the distribution of the replicated resources is independent of the distribution of the peer failures. This is true if the underlying DHT algorithm is based on the Chord protocol [106], or on Kademlia [63], because in these implementations the peers forming a replication group are determined by their peer identifier, which is a "random" value.

However, if the underlying DHT algorithm is based on Pastry [91] or Tapestry [117], then the replication groups are formed by peers that are close to each other in terms of the underlying network topology. The result of locality-aware replication is that peers being physically located in the same data center are likely to form a replication group. In case of a black out in the data center, the entire replication group is lost.

Therefore, the replication component should not be based on Pastry's or Tapestry's built-in replication groups, but it should be implemented independently of the underlying DHT topology. The potential performance improvement with locality-aware replication should be sacrificed for better reliability in case of blackouts.

Evaluation of Race Conditions

In the evaluation above, the focus was on the probability that an entire replication group is lost because of peer failures. However, the control infrastructure might be temporarily unstable even if no entire replication group fails. In the remainder of this section, these temporary anomalies are analyzed.

In case a peer becomes unavailable, queries for that peer are routed to the peer's successor in the replication group. However, it might happen that the original peer becomes available again shortly afterwards. This leads to ambiguous situations when responses are routed back to the root of the tree structure. It might happen that some child nodes report their responses to the peer holding a replicated node, while other child nodes report their responses to the original node.

There are two ways to deal with this situation. First, this situation can be avoided using consensus protocols to elect the responsible peer within the replication group. That way, the peers in the replication group always have a consistent view of the responsibilities. The disadvantage of this approach is that running these protocols requires a significant amount of messages being exchanged, causing a delay in routing operations.

The second way to deal with this situation is to ignore it. In that case, neither the original peer nor the peer in the replication group will receive responses from all children, so both peers will run into a timeout and forward the incomplete response. As a result, the two incomplete responses will finally reach the utility, and the application at the utility's site can put these responses together to reconstruct the complete response. Considering the low churn rates in the control infrastructure, this should happen very rarely, so this seems to be the best alternative to handle temporary anomalies.

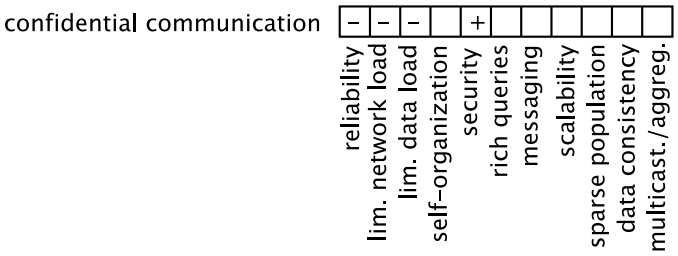
Summary, Conclusions and Outlook

In the section above, the evaluation of the routing and aggregation component was presented. The focus of the evaluation was on reliability. It was shown that combined with a replication component, the routing infrastructure provides a much higher level of reliability than classical client server systems.

Most of the results of the evaluation were found analytically. However, all of the results have been verified in simulation. The evaluation above answered the most important questions about the routing infrastructure. However, it is up to future research to set up a real-world implementation, allowing for direct experiments with real data from power generating devices.

7.3.4. Impact of the New Components on the Requirement Areas

In Section 4, three industrial application scenarios were introduced, and the application requirements were identified. In Section 5, state-of-the-art in peer-to-peer-based decentralization was reviewed, and functionality found in related work was generalized to domain independent service layer components. The



impact of these components on these requirement areas was recorded, using a '+' symbol for positive impact, and a '–' symbol for negative impact.

In this subsection, these records will be complemented with the impact of the new components presented above. The resulting map of service layer components will be discussed in the next section.

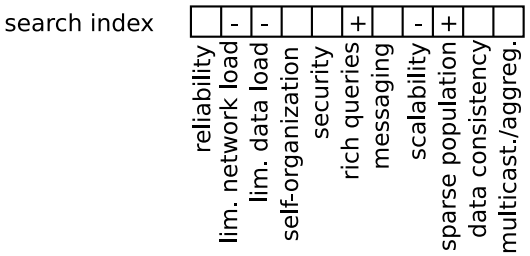
Confidential Business Communication

The figure above summarizes the impact of the confidential communication component on the requirements identified in Section 4.

The negative impact on *reliability* has been described in Section 6: The confidential communication component operates in a way that data resources are not published directly on the responsible peer, but resources are rather sent along a random path, veiling the original creator of the resource and its final destination. As a consequence, the creator of a resource hands the resource over to the first hop, having no way to verify if the resource will finally reach its destination. Therefore, applying the confidential communication component might decrease the level of reliability as compared to applications where resources can be published directly on the target peer. Reliable communication can be implemented using circuits, as described in Section 6, but this has not been included in the implementation described in this thesis. Rather, application layer acknowledgments have been proposed for the CPD scenario.

The effects on *limitations for network load* and on *security* show the trade-off to be made when implementing the component. Increasing the length of the random path results in a higher level of security but increases the network traffic. In the evaluation above, a path length of five was proposed as a reasonable value in the CPD scenario.

There is a negative impact on *limitations of data load*, as each peer needs to maintain a pool of IP addresses and public keys. However, in the CPD scenario this effect was negligible, as the amount of additional data to be stored is very small as compared to the size of the business documents. There is no effect on *self-organization*, as onion routing is implemented in a decentralized



way. *Scalability* is not affected, because the length of the routing path is constant and does not depend on the overall size of the overlay. For the other requirement areas: *rich queries*, *messaging*, *sparse population*, *data consistency*, and *multicasting and aggregation* no impact was observed.

Search Index

The EPHT algorithm presented in Section 6 enables the implementation of a new component providing a search index for decentralized user directories.

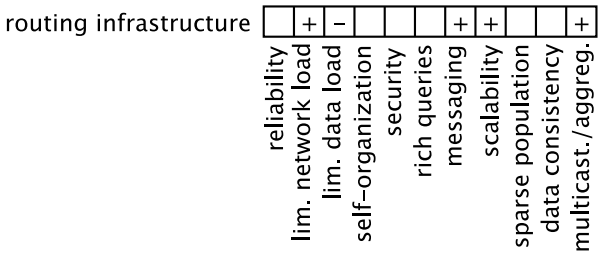
The search index does not increase or decrease the level of redundancy in the overlay, i.e. there is no impact on *reliability*. *Limitations of network load* and *limitations of data load* might both be violated when using a search index. The search index requires the peers to store a significant amount of additional data, and resolving a query using the search index results in a significant number of messages to be exchanged in the overlay. However, evaluation above showed that the search index complies with the requirements of the Peer Things project.

Self-organization is not affected, as the search index works in a fully decentralized way. There was no effect observed on *security* requirements.

Obviously, there is a positive impact on support for *rich queries*, as this is the primary goal of the search index component.

Messaging is not affected. The negative impact on *scalability* is because some queries may trigger interaction with a large part of the overlay, as for example querying all names starting with `'SCH*'` in a German phone book. Section 6 proposed to implement a limit for the maximum number of entries to be retrieved.

The search index component was specifically designed for *sparsely populated* overlays, maintaining a linked list skipping empty nodes. There was no effect on *data consistency* or *multicasting and aggregation*.



Routing and Aggregation Infrastructure

The evaluation of the distributed power generation scenario showed that a high level of *reliability* can be achieved when combining the routing and aggregation component with a replication component. However, the routing and aggregation component itself does not increase reliability.

The tree-shaped architecture of the routing and aggregation component has a positive impact on *limitations for network load*, as compared to regular singlecast solutions. The negative effect on *limitation for data load* is due to the fact that peers are required to store additional routing information.

Self-organization is not affected, as the routing and aggregation tree is maintained in a fully decentralized way. There were no *security* concerns with the component in the use case. *Rich queries* are not concerned.

The primary goal of designing the component was to create a scalable messaging infrastructure, which results in a positive effect on the requirements for *scalability* and *messaging*. The requirement areas *sparse population* and *data consistency* are not affected, as the routing and aggregation tree does not manage data in the overlay.

Obviously, the requirement for *multicasting and aggregation* can be fulfilled applying the routing and aggregation tree.

Summary of the Impact Map

Above, the map of service layer components started in Section 4 was complemented with the new components introduced in Section 6. A summarizing presentation including all components will be discussed in Section 8.

7.3.5. Summary

Above, the three service layer components introduced in Section 6 were evaluated. All of the components are related to application scenarios from current

Siemens research projects. The requirements for the evaluation were derived from the corresponding projects.

Three different approaches for evaluating software components were identified: direct experiments, analytical evaluation, and simulation. All of these approaches were taken in the evaluation above: The confidential communication component was implemented in a real-world framework that was deployed in a pilot project with FIAT and Siemens, which allowed direct experiments. The EPHT was mainly evaluated with simulation. The evaluation of the routing and aggregation tree was primarily based on mathematical combinatorics.

The contribution of the present thesis is twofold: Firstly, three service layer components are developed enabling peer-to-peer-based decentralization in application scenarios that used to be implemented in a centralized way. Secondly, an architecture for peer-to-peer-based applications is presented, and a methodology for applying peer-to-peer-based decentralization to new fields of application. The first part of this section dealt with the evaluation of the service layer components. Below, the architecture and methodology are evaluated.

7.4. Evaluation of the Architecture and Methodology

7.4.1. Criteria for Evaluation Software Architectures

When evaluating software systems, there are two types of criteria to be considered [32]: criteria that can be observed and measured at runtime like latency or reliability, and criteria that are not observable at runtime like modifiability or reusability.

The evaluation of the service layer components above is based on criteria that are observable at runtime. The corresponding requirements were defined in the respective Siemens projects. In contrast, the evaluation of the architecture and methodology cannot be expressed in terms of latency or throughput. Rather, soft criteria need to be considered for the evaluation.

When trying to find criteria suitable for evaluating an architecture, the following question needs to be answered: *What makes a "good" software architecture?* However, there is no architecture being good or bad in itself [32, 11]. Rather, an architecture can only be evaluated with respect to a given use case scenario.

The remainder of this section provides an evaluation of the architecture and methodology used in this thesis. The goal is to learn if the architecture and methodology performs better than related approaches when applied to the application scenarios considered in this thesis.

7.4.2. Review of the Architecture and Methodology

In Section 3, an architecture for decentralized, peer-to-peer-based applications was presented, introducing a component-based service layer between the raw overlay layer and the application layer. The service layer is composed of generic, domain independent components that can be used as building blocks to support application requirements. That way, application developers can benefit from the repetitive nature of application requirements observed in Section 4.

From this architecture, a methodology was derived for transferring current results in peer-to-peer-based decentralization to new fields of applications. The *gap analysis* relates the application requirements to the functionality offered by current service layer components. As a result, the gap between state-of-the-art and the application requirements becomes clear, and the requirements for new components are be identified.

Having analyzed that gap, new components can be implemented closing that gap. It becomes unnecessary to develop entire applications, which would require a lot of repetitive development effort. Moving the focus to the yet unsolved challenges is the major benefit of the methodology introduced in this thesis.

7.4.3. Evaluation

The evaluation of the architecture and methodology is organized by five attributes [32, 11]: Reusability, modifiability, integratability, testability, and portability.

Reusability. The architecture and methodology presented in this thesis was applied to three different application scenarios: Business collaboration, a scalable user directory, and distributed power generation. Although these scenarios are from different application domains, the requirements analysis in Section 4 showed that many requirements appear in more than one scenario in a similar way.

The modular architecture presented in this thesis exploits the repetitive nature of the requirements. Functionality found in related work is generalized as domain independent service layer components that can be reused in new fields of application, as shown in Section 5. That way, reusability of state-of-the-art results is enabled.

However, care must be taken, because some service layer components may have negative impact on other components' functionality. The interactivity among service layer components will be discussed in Section 8.

Modifiability. The component-based approach enables the modification of parts of an application, while other parts keep untouched. For example, replication is considered to be service layer functionality according to the architecture presented in Section 3. In order to use another replication strategy, the replication component needs to be replaced.

The evaluation of the routing and aggregation component showed that best reliability is achieved with replication that does not support locality awareness. Related peer-to-peer systems tend to include replication as built-in functionality in the overlay[13]. With built-in replication, high development effort is required for replacing the replication strategy. Using a modular replication component on the service layer, locality-independent replication can easily be enabled by switching to another service layer component.

Integratability. When designing the interface between the overlay and the service layer in Section 3, it was decided to put as much functionality as possible into the service layer, leaving the overlay as a lightweight routing infrastructure. This approach minimizes the interrelationships between the service layer and the overlay, and enables the integration of functionality on the service layer that would require modifications on the internal overlay data structures in most other architectures.

An example for this can be found in the evaluation of the confidential communication component. Sending a resource along a random path requires access to the raw routing functionality of the underlying peer-to-peer overlay. If the overlay provided a higher level CRUD[14]-like API, the implementation of random paths would interweave with the internal API of the overlay.

Testability. The service layer components each implement functional units, and these units can be tested independently of each other. However, some components have mutual impact on each other, which means that some components may change their behavior when integrated in the overall application. For example, the use of caching would compromise the functionality of a data consistency component.

In Section 5 and Section 7, the impact of the components on the requirement areas was recorded. These records will be merged in Section 8, showing the mutual impact of the components on each other. The resulting presentation helps application developers to decide which components

[13] See the Tapestry review in Section 1.3.
[14] CRUD: Create, Read, Update, and Delete.

can be tested on their own, and which components may change their behavior when integrated in the overall application.

Portability. The architecture and methodology presented in this thesis was applied to three application scenarios covering a wide range of different application domains. Therefore, it becomes evident that the architecture and methodology can probably be used in other application scenarios as well.

7.4.4. Summary of the Evaluation of the Architecture and Methodology

Above, five properties were presented that can be found in related work on evaluating software architectures [32]: Reusability, modifiability, integratability, testability, and portability. These properties represent soft criteria for evaluating software architectures. The goal of the evaluation is to determine if the architecture and methodology is better than related approaches when applied to the application scenarios targeted in this thesis. It is not possible to evaluate a software architecture independently of an application scenario [11].

For each of the five evaluation criteria, the experiences made in this thesis were described. It turned out that the architecture and methodology has advantages as compared to approaches that do not use a service layer. The component-based approach helps application developers to transfer current research results to new application scenarios, and to focus on the gap between state-of-the-art and the application requirements.

As the application scenarios are chosen from a wide range of different domains, it is likely that the architecture and methodology will be useful in future applications as well.

7.5. Summary

The evaluation presented above is split into two parts. Firstly, the service layer components introduced in Section 6 were evaluated. Secondly, an evaluation of the architecture and methodology of the service layer approach was presented. The results can be summarized as follows:

1. The evaluation of the service layer components proves that the application requirements can be fulfilled. The gap between state-of-the-art and the application requirements is closed. Using these components, the applications presented in Section 4 can be implemented in a decentralized way.

2. In the evaluation of the architecture and methodology, the worth and significance of the service layer approach was determined. This approach was taken consistently through all of the use case scenarios, and the experiences made with the different applications were taken as a basis for the evaluation.

   The architecture and methodology used in this thesis proved more suitable for the application scenarios than related approaches that do not use a service layer. The service layer allows a methodology providing a generic way to identify the gap between state-of-the-art research and application requirements. The gap analysis helped to keep the focus on the yet unsolved challenges, reducing the development time for the new applications.

In the next section, a map of service layer components will be presented summarizing the impact of the components on the requirements.

8. Conclusions & Future Work

8.1. Contributions of this Thesis

The goal of this thesis was to make peer-to-peer-based decentralization available for industrial applications. The contribution of this thesis is twofold:

1. An architecture and a methodology were presented that help application developers from industry to transfer state-of-the-art solutions in peer-to-peer-based decentralization into their application domains. That way, functionality from related work can be re-used, and developers can focus on the specific requirements that have not yet been addressed in related work.

2. The methodology was applied to three industrial application scenarios. For each of these scenarios a service layer component was developed closing the gap between state-of-the-art solutions and the application requirements. These components enable the use of peer-to-peer-based decentralization in new application domains.

In the following two subsections, the results will be summarized.

8.1.1. Architecture and Methodology

With the rise of peer-to-peer computing, a considerable amount of research results on decentralized distributed systems has become available. The self-organization and resilience provided by these systems offer new opportunities for applying decentralization in industrial applications. However, there are still very few industrial applications benefiting from these results. Many peer-to-peer infrastructures are monolithic systems, designed with a specific use case in mind. This makes it difficult for application developers from industry to transfer these systems to new fields of application.

The goal of this thesis was to enable industrial applications to benefit from peer-to-peer-based decentralization. In this thesis, three industrial application scenarios for decentralized systems were analyzed: the decentralized business collaboration scenario from the automotive industry, the user directory for

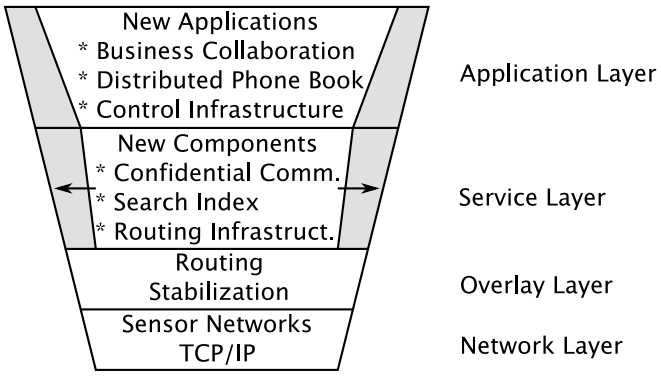


Figure 8.1.: New Service Layer Components Enabling New Applications

a decentralized communication platform, and the control infrastructure for distributed power generation. The main outcome of the analysis was: Although the scenarios are taken from a wide range of application domains, many requirements appear in more than one scenario in a similar way.

An architecture for peer-to-peer-based decentralized applications was introduced that makes use of the repetitive nature of the requirements. The architecture introduces a component-based service layer, providing application-specific functionality on top of a generic peer-to-peer overlay. The peer-to-peer overlay is reduced to basic routing and stabilization functionality. This functionality is complemented by modular, domain independent service layer components, providing replication, access control, data consistency, messaging, etc.

Based on this architecture, a methodology was presented helping developers to identify the gap between application requirements and state-of-the-art service layer components. Applying this methodology decreases development time for new applications, as it helps the developer to re-use existing functionality and to focus on open challenges.

8.1.2. Service Layer Components

The methodology was applied to three industrial application scenarios. For each of these scenarios, a new service layer component was developed closing the gap between state-of-the-art solutions and the application requirements:

Confidential Communication. Security in peer-to-peer networks is a relatively new topic. The gap analysis for the business collaboration scenario in Section 6 showed that related work does not provide any confidential publish/subscribe functionality for DHTs. Therefore, a new service layer

component was developed that provides confidentiality. The component is based on the onion routing algorithm, which has originally been proposed for anonymous email communication. The confidential communication component adapts this algorithm for the DHT-based collaboration platform, and introduces DHT-specific features, e.g. a pool of peer identifiers and public keys that is used to establish the random path. The confidential communication component closes the gap between state-of-the-art peer-to-peer functionality and the requirements of the business collaboration platform, and thus enables the use of peer-to-peer-based decentralization in the business collaboration scenario.

Search Index. The gap analysis for the distributed user directory in Section 6 showed that Zipf-distributed user data cannot be stored and searched efficiently in DHTs using approaches found in related work. Within the Peer Things project, related range query mechanisms like Squid, Mercury, and Skip Graphs have been evaluated, but it turned out that none of them fulfills the requirements in the user directory scenario [55]. Therefore, a new component has been developed enabling range queries in the user directory. The component shares ideas with the Prefix Hash Tree proposed in [82], but differs from the Prefix Hash Tree in several ways, e.g. it is not limitted to a binary structure, supports combination with a caching component, and uses concatenation for multi-dimensional keywords instead of SFCs. The evaluation in Section 7 shows that the component fulfills the requirements of the Peer Things project and therefore enables the implementation of a fully decentralized user directory for Peer Things.

Routing and Aggregation. The third scenario looked at in this thesis was distributed power generation. The gap analysis in Section 6 showed that most requirements can be fulfilled using current research results. However, there is no related work on routing and aggregating sensor and control data. A new component was presented providing routing and aggregation functionality. The component implements an XML-like tree structure, but differs from related work on distributed XML-processing, as related approaches tend to focus on distributed data stores, as opposed to routing and aggregation functionality. The contributions of the routing and aggregation component is twofold:

Firstly, the chunk-based approach presented in this thesis is more efficient than related approaches, and it relieves the traffic load of the root node. Secondly, the stateless approach enables more reliable routing and the implementation of advanced features, like locality-aware routing.

The evaluation in Section 7 showed that the routing and aggregation

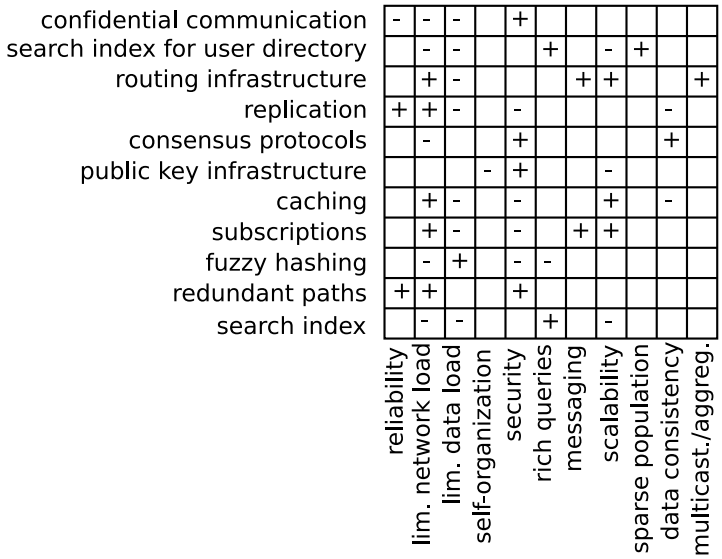


Figure 8.2.: Map of Service Layer Components

component provides a high level of reliability, and fulfills the requirements of the distributed power generation scenario.

The evaluation in Section 7 showed that these components enable the use of peer-to-peer-based decentralization in the corresponding Siemens projects. However, the components are not limited to these applications. Rather, they provide generic functionality and can be re-used in other application scenarios as well.

In the remainder of this section, experiences made with the service layer components will be brought together in a map, showing the mutual impact of the components.

8.2. Interactivity Map of the Service Layer

Figure 8.2 shows a map of service layer components, summarizing their impact on the requirement areas identified in Section 5 and 7.

8.2.1. Interpretation for Application Developers

As part of this thesis, a methodology was presented and applied that helps developers from industry to transfer state-of-the-art in peer-to-peer-based decen-

tralization to their fields of application. The map of service layer components indicates which components are expected to work together without difficulties, and which combinations are likely to have negative effects on the component's functionality.

1. Two service layer components are **loosely coupled** if they do not impact the same requirement area. In that case, the components are likely to have no negative impact on each other. However, the map does only show effects that have been observed in the work considered in this thesis. There might be additional conflicts in other application scenarios.

2. When two components have impact on the same requirement area, these components are **tightly coupled**. Developers need to focus on the challenges arising when these components are used together. The map does not provide further information on what specific problems are to be expected. If components have contradicting effects, trade-offs have to be made depending on the use case scenario.

An example of components that have high impact on each other can be found in the CPD scenario presented in Section 4. Here, the requirements for security and limited network load are contradicting, and a trade-off needed to be found providing a reasonable level of security while keeping the network load within the limits.

If the application requirements are so strict that no trade-off can be found, then peer-to-peer-based decentralization should not be used for that application, and centralized entities need to be introduced.

In conclusion, the map does not relieve application developers from evaluating new applications. However, the map points out potential pitfalls, and helps developers to focus on the crucial points of their applications.

8.2.2. Interpretation Regarding Peer-to-Peer in General

The previous subsection showed that the map of service layer components may help application developers to identify the crucial points when designing their application-specific service layer. In this subsection, some general conclusions on the strengths and weaknesses of peer-to-peer-based decentralization are presented.

Firstly, there is no one-to-one mapping of service layer components and requirements. All service layer components have impact on several requirement areas. All service layer components have positive and negative impact. While some requirements are fulfilled using these components, other requirements

may be compromised. This means that applying any of the service layer components is always a trade-off.

Secondly, the map indicates the weaknesses of peer-to-peer-based decentralization. As shown in Figure 8.2, the requirement for data load limitations has more '–' symbols than any other requirement. This means that peer-to-peer-based decentralization is limited if a large amount of data needs to be stored in a decentralized way[1]. This is the reason why there is so much related work targeting load balancing and hot spots.

Thirdly, in the horizontal direction the components with the most '–' symbols can be identified, e.g. confidential communication or fuzzy hashing. These components must be used with care, and difficulties are to be expected when functionality provided by one of these components is required by an application.

8.2.3. Summary

The interactivity map presented above summarizes the service layer components' impact on the application requirements identified in this thesis. The map shows experiences made in this work, and draws the attention of application developers at crucial points of their applications.

However, the map does only show conflicts that have been observed in related applications. New applications may involve new conflicts of service layer components. Therefore, the map does not relieve developers from evaluating their applications. However, the presentation helps them to focus on the potential pitfalls first.

8.3. Future Work

In this thesis, a component-based architecture for decentralized applications was presented. A methodology has been derived from this architecture helping application developers to identify the gap between state-of-the-art solutions and application requirements, and to transfer current research results in peer-to-peer-based decentralization to new fields of application. This methodology was applied to three industrial application scenarios, and new components were developed closing that gap.

This thesis leaves several areas to future work. Firstly, the three new service layer components presented in this thesis allow more applications to benefit

[1] Note that typical file sharing applications do not distribute content in a decentralized way. Instead, each peer acts as a central server sharing its own content.

from peer-to-peer-based decentralization, and there is future work related to each of these components. Secondly, the architecture and the methodology presented in this thesis provide a starting point for further standardization and unification of peer-to-peer-based applications. In the remainder of this section, future research directions will be presented.

8.3.1. The Components

Confidentiality in Decentralized Business Collaboration

As the first application scenario, a decentralized business collaboration platform was introduced, supporting the Collaborative Product Development (CPD) process in the automotive industry. A service layer component based on Onion Routing was developed enabling confidential communication within this platform.

The confidential communication component successfully defeats the security threats identified in Section 6. However, security is a moving target, and new threats will arise in the future. The confidential communication component provides a first step towards secure DHTs, but more security enhancements will become necessary when new attack scenarios appear.

Security in peer-to-peer-based decentralization is a relatively new research topic. The development of the confidential communication component showed that security issues can be met successfully. There will be more research targeting secure DHTs in the future. When more research becomes available, the different approaches can be compared and a common methodology for security assessment in decentralized applications can be derived.

Search Index for the Decentralized User Directory

The second application scenario presented in this thesis was a decentralized user directory to be integrated into the Peer Things communication platform. A search index component was presented enabling range queries in the user directory.

Due to the required scalability of 500 000 users and more, the component was evaluated in a simulation environment, using a high level of abstraction. The underlying peer-to-peer-infrastructure was modeled as a stable hash table. Although the assumptions have been made with care, experiments in real world deployments are necessary to verify the simulation results.

As shown in the related work in Section 5, there are many other publications targeting search indexes on top of DHTs. Each of the approaches has its own strengths and weaknesses. It is up to future work to provide a concise

taxonomy of decentralized search components, helping application developers to learn which algorithm suits best for their specific use case requirements.

Routing and Aggregation Infrastructure for Distributed Power Generation

The third and last application scenario introduced in this thesis was a control infrastructure for distributed power generation. A service layer component was presented allowing for routing and aggregation of sensor and control data in a decentralized way.

In this thesis, the focus of the evaluation was on the level of reliability. It was shown that the peer-to-peer-based approach provides a higher level of reliability than client-server-based architectures, even if peers are run on unreliable hardware.

However, other aspects are to be addressed as soon as real-world implementations become available. For example, latency of the control commands needs to be measured in the Internet. Moreover, the shift towards decentralization in power generation is relatively new, and the impact on power distribution networks is still to be evaluated.

As opposed to the other two application scenarios, decentralized power generation is a future scenario, and many aspects cannot be foreseen today. However, this work showed that peer-to-peer-based decentralization is a promising way for implementing control infrastructures. The applicability of the results needs to be evaluated as soon as utilities start deploying real-world control for generators operated by end users.

8.3.2. The Architecture and Methodology

Above, future work was presented regarding the three service layer components introduced in Section 6. The remainder of this section shows future research directions related to the architecture and methodology.

Standardized Assessment of Service Layer Components

The presentation of the mutual impact of the components in Figure 8.2 shows tendencies rather than hard evaluation criteria. It would be desirable to find a generic approach for quantifying the components' impact. The quantification could be done by counting the '+' and '–' symbols, weighting the impact with respect to the relevance of the requirement.

However, the requirements' relevance is hard to determine. Firstly, it is application specific. The same requirement might be of minor importance for

some application, while it is crucial for another application. Secondly, the requirements' relevance depends on the stakeholder. A security officer would certainly consider security to be the most relevant requirement, while an end user might consider low latency as more important.

Therefore, it will be difficult to find a methodology for assessing service layer components that goes beyond the simple '+' and '–' symbols used in this thesis.

Standardization of Interfaces

The modular approach presented in this thesis stays on a conceptual level. This thesis did not define an API for service layer components. Therefore, on the technical level, it is likely that code needs to be adapted when a component is re-used in a new application.

It is up to future work to develop a framework defining standard interfaces for service layer components. This framework could be inspired by the SOA, or by related projects like Kamaelia[2].

Development Support for Service Layer Components

Today, software development makes frequent use of frameworks and code generation tools. For example, in order to implement a service oriented application, one would use an application server to run the services, a persistence framework to store objects in a database, etc. Moreover, the stubs for the service implementations would be generated automatically, e.g. from WSDL definitions.

Starting from the component-based architecture presented in this thesis, the vision is to develop a full-featured framework for designing decentralized infrastructures. Ideally, the developer would simply define the requirements to be fulfilled, and the framework would generate an application-specific infrastructure.

However, a large amount of future research is necessary to implement such a framework.

[2] http://kamaelia.sourceforge.net/Home, 7 November 2007

A. Current DHT Implementations

As defined in Section 3, the operations provided on the overlay layer are routing and stabilization. This section introduces the most influential DHT algorithms, and shows how they implement this functionality.

The purpose of this section is to introduce a common terminology that can be used to describe the functionality of the overlay layer independently of the actual peer-to-peer algorithm. More detailed surveys of different DHT algorithms can be found in related publications [60, 62].

The DHTs introduced here are: CAN, Chord, Pastry, Tapestry, and Kademlia. The first four DHTs are chosen because they were the first DHTs to be published. Kademlia is added because it is the most widely used DHT. The structure of the presentations is as follows:

Topology gives an overview of how the overlay is structured.

Routing explains how identifiers are resolved and responsible peers are located.

Join and Stabilization explains the mechanism handling peers that leave or enter the overlay.

Interface to the Service Layer summarizes the functionality provided to the upper layer.

In the remainder of this section, the following variables are used:

- n is the number of peers.
- m is the number of Bits per identifier. A frequently used value in current peer-to-peer applications is $m = 160$.
- d is the number of identifiers per query. This is also called the dimension of the **identifier space**.

As each peer must be associated with at least one identifier, it is clear that $n \leq 2^{dm}$.

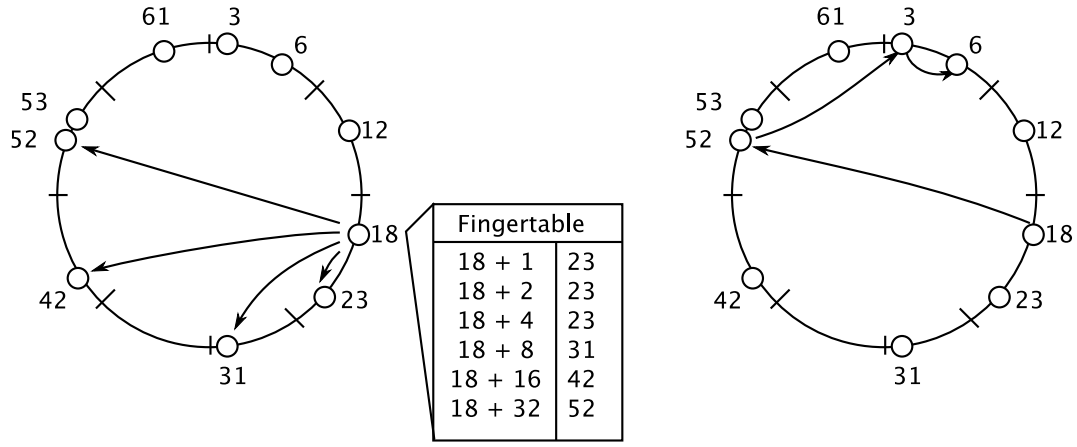


Figure A.1.: Chord Fingertable (left) and Lookup Example (right).

A.1. The Chord DHT

Chord [106] has been among the earliest DHT algorithms to be published. Chord's topology and lookup procedure suits well for being illustrated graphically, which is one reason why Chord is frequently used as a reference in DHT-related publications. Chord is developed and maintained at the Massachusetts Institute of Technology.

Topology. In Chord, peers are ordered in a ring structure modulo 2^m, according to their identifiers. Each peer is responsible for the identifier range between its predecessor and itself. Each peer knows its successor and predecessor in the ring. An example of a Chord ring with $m = 6$ is illustrated in Figure A.1.

Lookup. As each peer knows its successor, queries could be resolved in a way that each peer simply forwards the query to its successor until the responsible peer is reached. This would result in a lookup performance in $O(n)$. In order to implement lookups more efficiently, each peer stores a **finger table**. The finger table is a list of peers that can be used as shortcuts when traversing the ring. A peer with identifier i stores the peers being responsible for the identifiers $i + 1, i + 2, i + 4, \ldots, i + 2^{m/2}$ modulo 2^m. To exemplify this, Figure A.1 on the left shows the finger table of the peer with identifier 18.

When a lookup operation is performed, the peer forwards the query to the closest known peer in the finger table. That peer handles the query recursively. This results in $O(\log_2 n)$ messages to be sent in order to resolve one query. An

example for a lookup operation is shown on the right hand side in Figure A.1. In the example, the peer with identifier 18 queries the identifier 4. The peer with identifier 6 is responsible for that identifier, and the query is recursively forwarded to the peer with identifier 6.

Join and Stabilization. In order to join a Chord ring, a peer needs to know at least one peer that is already a member of the ring. This initially known peer is called the **rendez-vous**. Any peer may act as the rendez-vous for new peers joining the ring. The rendez-vous is asked to look up the new peer's successor in the ring. The new peer asks its successor for its predecessor, and informs both peers about its arrival.

Stabilization is used to keep the successor, predecessor and finger table entries updated when peers join or leave the Chord ring. When a peer asks its successor for its predecessor, the result should be the peer itself. If the result is a different peer, then a new peer joined the network and settled down as the new successor. In order to keep the infrastructure stable, each peer asks periodically for its successor's predecessor. These periodic messages are used in two ways: Firstly, the peer querying for its successor's predecessor learns about new peers, secondly, the successor receiving a message learns about new predecessors.

In addition to that, each peer periodically looks up its 2^ith successor, and updates the corresponding entry in the finger table.

Interface for Service Layer Components. In Chord, there is a one-to-one mapping between queried identifiers and responsible peers. Each query must contain exactly one identifier to be queried, and the result is exactly one peer being currently responsible for this identifier.

A.2. The CAN DHT

Among the early DHTs, the **Content Addressable Network (CAN)** [84] is the only one that is designed to handle multiple identifiers per query. However, the lookup performance in CAN is lower than the lookup performance of the other algorithms presented here, which is one of the reasons why CAN is not very popular in current research on peer-to-peer-based decentralization. CAN was developed at the UC Berkeley.

Topology. CAN maintains a d-dimensional identifier cube, where each axis represents one identifier range. Figure A.2 shows a two-dimensional CAN

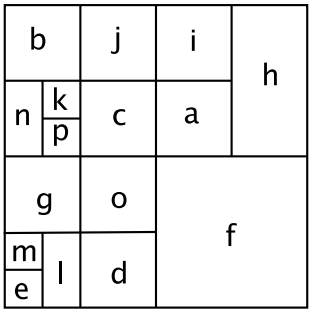


Figure A.2.: Example of a 2-Dimensional CAN Topology

topology. The letters *a* through *p* represent peers being located in the two-dimensional identifier space.

Lookup. As illustrated in Figure A.2, each peer is responsible for a certain box in the d-dimensional CAN cube. Each peer knows its neighboring peers, i.e. the peers sharing a common borderline. For example, in Figure A.2 the neighbors of peer *c* are *j*, *a*, *o*, *k*, and *p*. Queries are recursively forwarded to one of the neighbors being closer to the queried identifier. The number of hops being necessary to resolve a query is in $O(\sqrt[d]{n})$. This means that CAN performs better the more dimensions are used for the identifiers. In the worst case with $d = 1$, CAN's lookup operation is in $O(n)$, which results in significantly higher latency than Chord's $O(\log_2 n)$.

Join and Stabilization. As in Chord, new peers must know a rendez-vous, which is an arbitrary peer that is already part of the overlay. The new peer uses the rendez-vous to look up its identifier, and informs the peer currently being responsible for that identifier. Then, the box of the current peer splits into two halves, and the new peer takes over responsibility for one of these halves. The list of neighbors for the new peer is initialized with the list from the old peer, and all neighbors are informed about the change.

Each peer sends periodic update messages to each of its neighbors, including the list of their neighbors. The prolonged absence of these messages triggers the neighbors to issue a take-over message to all of the absent peer's neighbors, indicating that the neighbor is willing to take over responsibility for the absent peer's box. The peer that sent the first take-over message will take over responsibility for the box.

Interface for Service Layer Components. CAN is the only DHT presented here that natively supports $d > 1$, i.e. queries for more than one identifier. In other DHTs, like Chord, it is necessary to implement appropriate service layer components if an application requires support for multiple identifiers. These service layer components are unnecessary if CAN is used as the underlying peer-to-peer overlay.

A.3. The Pastry and Tapestry DHTs

Pastry [91] shares similarities with Chord, but introduces several new ideas that make it more flexible than Chord. Pastry is quite popular in current research, not least because there are mature open source implementation available, such as the Bamboo DHT [87], or Free Pastry[1]. Pastry was developed at Microsoft Research.

Pastry is very similar to **Tapestry** [117], which was developed at the UC Berkeley. Both DHTs are based on the routing algorithm by Plaxton, Rajaraman, and Richa [72]. This presentation describes the Pastry DHT, but the discussion presented here is valid for Tapestry as well.

Topology. As in Chord, Pastry peers are located in a circular topology which ranges from 0 to $2^m - 1$. But unlike Chord, Pastry interprets identifiers as sequences of hexadecimal digits[2]. The lookup mechanism is constructed with respect to this sequence of digits.

Lookup. As in Chord, each Pastry peer maintains a finger table in order to improve lookup performance. But unlike Chord peers, Pastry peers choose 16 fingers for each prefix of the hexadecimal representation of their identifiers. Peer identifiers are strings of $m/4$ hexadecimal digits. The finger table contains $m/4$ rows, each of which representing a prefix of the peer's identifier. For each prefix, there are 16 columns for the 16 possible ways how the next digit could look like. The finger table contains one peer being responsible for each of these 16 possible prefixes. An example for a Pastry finger table is shown in Figure A.3.

When an identifier is queried, the identifier is processed digit by digit. A peer having the first i digits in common with the queried identifier is able to forward

[1] http://www.freepastry.org, 6 February 2008

[2] The Pastry publication keeps this more general, and says that identifiers are interpreted as sequences of digits with base 2^b, where $b = 4$ is a typical value.

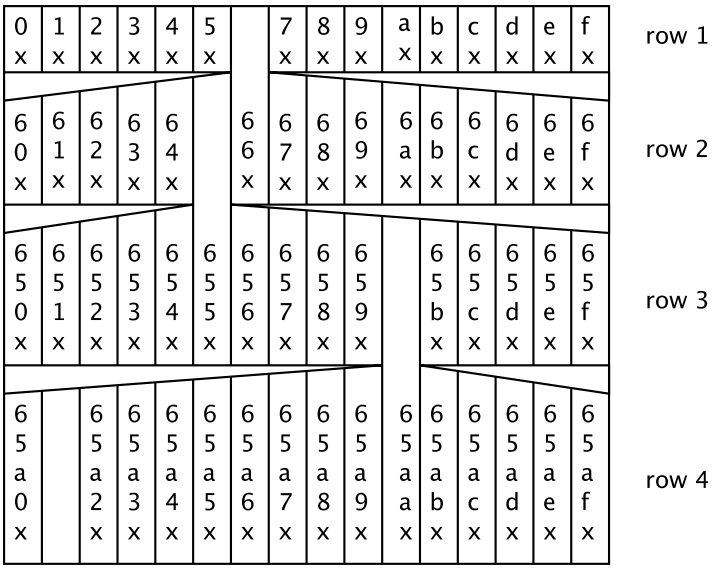


Figure A.3.: Example of a finger table of a Pastry peer with identifier 65a1*x*, where *x* represents an arbitrary suffix [19]

the query to a peer having the first $i + 1$ digits in common. Therefore, lookup performance is $O(\log_{2^4} n)$.

However, although the performance seems more efficient than Chord's performance of $O(\log_2 n)$, there is no substantial difference between the performances of both algorithms. Chord could be easily extended to perform lookups within $O(\log_i n)$, where i is an arbitrary integer. The only change necessary is to let Chord's fingers divide the ring in i parts instead of 2 parts. Figure A.4 illustrates a Chord ring where the fingers divide the range in 3 parts, which would result in a routing performance of $O(\log_3 n)$.

Join and Stabilization. As with the other DHTs, a rendez-vous must be known to a new peer joining the overlay. The rendez-vous is asked to send a lookup message for the identifier of the new peer. All peers on the lookup path send their finger tables to the new peer, and all peers that need to be aware of the arrival are informed.

Interface for Service Layer Components. Pastry basically provides the same interface as Chord, i.e. a queried identifier is matched to the peer being responsible for that identifier. However, Pastry provides some additional

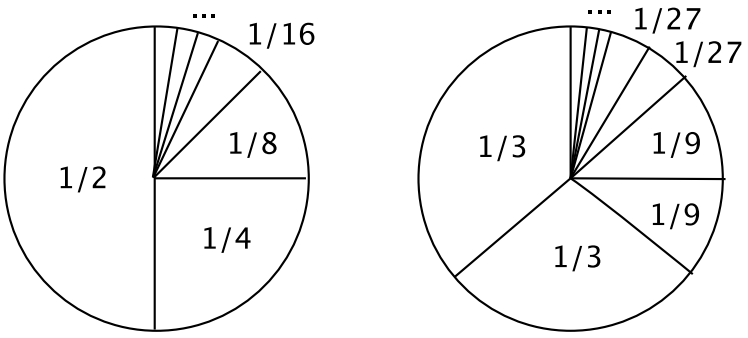


Figure A.4.: Illustration of Chord's finger tables. On the left side there is a regular Chord with routing performance $O(\log_2 n)$, on the right side there is a modified Chord ring with routing performance $O(\log_3 n)$.

flexibility. The finger table is constructed in a way that each entry is one of the peers being responsible for the given prefix. This means there are several peers that would fit for each entry in the fingertable. The choice of these peers can be done by service layer components. The original Pastry publication describes the Vivaldi algorithm that chooses peers being close in terms of network latency.

A.4. The Kademlia DHT

Kademlia [63] has been developed at the University of New York in 2002, one year later than the other DHTs described here. Nowadays, Kademlia is the most widely used DHT algorithm in the Internet, because it is applied in the Kad Network, which is part of eDonkey, which is one of the most popular filesharing networks today.

Topology. Unlike Chord, Kademlia does not have a ring-shaped topology. Rather, the peers are structured in a binary tree, ordered by the Bit-wise representation of their peer identifier. Figure A.5 shows a Kademlia tree with four peers, 'a' through 'd'. In the example, the number of Bits per identifier $m = 3$.

The responsibility of the peers is determined by Kademlia's XOR metric. An example for this is shown in Figure A.6. If identifier 010 was looked up in the Kademlia overlay illustrated in Figure A.5, then peer 'd' would be closest to that identifier.

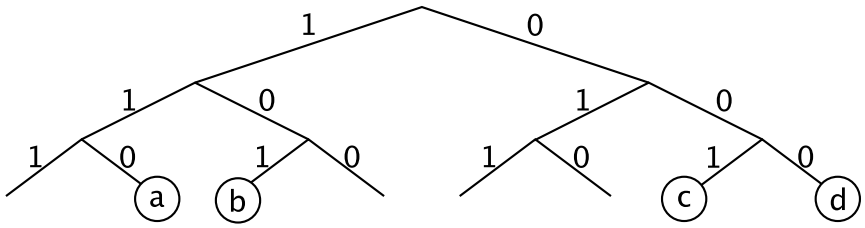


Figure A.5.: Kademlia's Binary Tree

In the graphical tree representation, the responsible peer is always located in the same sub-tree as the resource. If the sub-tree is empty, the next bigger sub-tree needs to be considered.

Lookup. As explained above, the peer begin responsible for an identifier is always located in the same sub-tree as the identifier itself. Sub-trees are determined by the prefix, i.e. the path to the sub-tree in Kademlia's tree structure.

Kademlia's finger tables are basically built in the same way as in Pastry, but the identifiers are processed Bit-wise and not hex-digit-wise. For each prefix length in $[0...m-1]$, each peer has a finger table entry that starts with the same prefix, but where the next binary digit differs. For example, peer c has the identifier 001 in Figure A.5. Its finger table would be built as follows:

Prefix Length 0. Peer c knows one peer with prefix 1.

Prefix Length 1. Peer c knows one peer with prefix 01.

Prefix Length 2. Peer c knows one peer with prefix 000.

The lookup is routed Bit by Bit, which results in a lookup performance of $O(\log_2 m)$.

As in Pastry, there is more than one possibility which peer to store at each position of the finger table. Kademlia introduces j-buckets and keeps j different entries per finger.

Join and Stabilization. the peers in the j-buckets are ordered according to uptime. As in Pastry, a new peer asks a rendez-vous to look up its identifier. Each peer on the path of the lookup route learns about the existence of the new peer.

peer	resource id ⊗ peer id		distance
a	010 ⊗ 110	=	100
b	010 ⊗ 101	=	111
c	010 ⊗ 001	=	011
d	010 ⊗ 000	=	010

Figure A.6.: Example of Kademlia's XOR Metric: Distance of Identifier 010

Interface for Service Layer Components. Like Chord, Pastry, and Tapestry, Kademlia provides a one-to-one mapping of a single identifier to a unique peer being closest to that identifier.

A.5. Summary

This appendix introduced the most influential DHT algorithms. The presentations were kept very briefly, more detailed descriptions in the algorithms can be found in the referenced publications. This appendix focused on the routing and stabilization functionality, the original publications may include more functionality in the overlay, like locality-awareness, replication, etc.

The goal of this appendix was to show that all DHTs provide a mapping of identifiers to peers currently being responsible for these identifiers. In most DHTs, this mapping is one-to-one. Except for CAN, all DHTs presented here have a lookup performance in $O(\log_i n)$, where n is the number of peers, and i depends on the design of the finger table, with $i = 2$ as a typical value.

Glossary

Analytical evaluation. A way of validating software systems. An abstract model of reality is established, and the software behavior validated mathematically with respect to this model. Analytical evaluation should be used if direct experiments in the real world are not feasible, page 108

Application layer. Implements distributed applications on top of an application-specific, decentralized service layer, page 28

ATHENA. Advanced Technologies for interoperability of Heterogeneous Enterprise Networks and their Applications; an Integrated Project funded by the European Commission aiming at interoperability of enterprise systems and applications, page 34

Authentication. The assurance that the originator of data resources or messages is who he claims to be, page 55

Axis. An implementation of the SOAP protocol in Java. On the server side, Axis can be used to implement Web services. On the client side, it can be used to call Web services, page 117

Back-route list. In the routing and aggregation component, a back route list is sent along with each query. That way, responses can be routed independently of queries, which enables locality-aware routing and increases the level of reliability, page 101

Business resource management framework (BRMF). A peer-to-peer-based, decentralized business collaboration platform, implemented at Siemens as part of the ATHENA project. The BRMF is based on the RMF, page 116

Castor. Library implementing a mapping between XML data and Java objects, page 117

Centralized peer-to-peer overlay. A Napster-like architecture where a central index-server keeps track of the peers and their data, page 21

Certificate authority (CA). Issues digital certificates in a PKI, page 56

Chord. A DHT developed at the Massachusetts Institute of Technology, page 154

Churn. In peer-to-peer terminology, the continuous appearance and disappearance of peers is called churn, page 29

Clean node failure. The most simple failure model, where the failing entity disappears, but does not send faulty messages, page 128

Collaborative product design (CPD). Business strategy where new products are developed in a collaborative process involving the OEM and its suppliers, page 36

Collaborative product development. See collaborative product design, page 36

Confidentiality. The assurance that the contents of data resources or messages can be read only by a particular group of recipients, page 56

Constructivistic. Constructivism is a perspective in philosophy that accepts theoretical reasoning as the only source of knowledge, page 7

Content addressable network (CAN). A DHT developed at the University of California at Berkeley, page 155

Coopetitive. Made-up word, composed of cooperative and competitive. Describes business processes where the participants are cooperators and competitors at the same time, page 56

Decentralized object location and routing (DOLR). A data-store-like API for DHTs. Unlike CAST, which is a multicast-streaming-like API for DHTs, page 5

Direct experiments. The preferred way for evaluating software components, as direct experiments in the real world do not rely on a simplifying model of reality, page 108

Distributed hash table (DHT). In this thesis, DHT is used interchangeably with structured peer-to-peer overlay, page 22

Distributed system. Summarizes all areas of computer science where two or more entities communicate with each other over a network, page 11

Empiricistic. Empirism is a perspective in philosophy that emphasizes the role of experience, page 7

Event-driven collaboration paradigm. A business collaboration paradigm emphasizing the reaction upon events, page 35

Extended prefix hash tree (EPHT). A search index for DHTs, enabling range queries in the decentralized user directory, page 81

Finger table. List of peer identifiers and IP addresses, enabling efficient location of peers in a DHT, page 154

Fuzzy hashing. Generic term, characterizing all load balancing techniques relying on replacing strict hashing by adaptive hashing, page 62

Globus toolkit. A community-based, open-architecture, open source set of services and software libraries that support Grids and Grid applications [37, p. 55], page 15

Gnutella. An unstructured peer-to-peer overlay, page 22

Grid. A new infrastructure that builds on today's Internet and Web to enable and exploits large-scale sharing of resources within distributed, often loosely coordinated groups what are sometimes termed *virtual organizations* [35], page 15

Hot spot. Overloaded peer. There are hot spots in terms of data load, and hot spots in terms of network traffic load, page 61

Identifier space. The set of identifiers. In most DHTs, the identifier space is one-dimensional. However, CAN allows multi-dimensional identifiers, page 153

Identifier. Unique value to be mapped to a peer in a DHT. In many applications, the identifiers are hash values of the resources to be stored, page 29

Integrity. The assurance that data resources or messages have not been altered intentionally or unintentionally, page 55

Interception handler. Internal API within the RMF, allows to implement hooks handling incoming messages, page 119

JXTA. A hybrid peer-to-peer overlay developed at SUN, page 22

Kademlia. An DHT developed at the University of New York, page 159

Layers. Architectural style for software systems, utilized for hiding implementation details. Each layer uses the functionality of the underlying layer and provides functionality to the upper layer, page 12

Locality-preserving hashing. Method for mapping multi-dimensional identifiers to one-dimensional identifiers, such that adjacent points in the multi-dimensional range become close to each other in the one-dimensional range, page 88

Loosely coupled. Two components are loosely coupled, if they have only little interaction with each other, page 147

Message-level simulator. Overlay simulator abstracting the underlying TCP/IP network, but simulating the peer-to-peer protocol messages on the overlay layer, page 110

Mockup transport. Internal functionality in the RMF, allowing to replace network communication with simulated network communication in order to perform simulation runs, page 110

Network layer. Implements TCP/IP or Sensor Networks. The network layer makes sure that each peer is addressable by each other peer, page 28

Networked organization. Independent people and groups act as independent nodes, link across boundaries, to work together for a common purpose; it has multiple leaders, lots of voluntary links and interacting levels[3], page 35

Onion routing. An algorithm enabling anonymous communication in distributed systems, page 74

Original equipment manufacturer (OEM). A company that produces a product, possibly using parts produced by its suppliers, page 36

Overlay layer. Implements the peer-to-peer overlay network, establishing a logical addressing scheme that is independent of the underlying hardware, page 28

P2psim. An overlay simulator developed at the Massachusetts Institute of Technology, page 111

Pastry. A DHT developed at Microsoft, page 157

Path expression. In the routing and aggregation infrastructure presented in Section 6, a path expression is used to unambiguously address nodes in the tree structure, page 96

[3] http://www.skyrme.com/insights/1netorg.htm, 12 February 2008

Peer pool. Cache of peer identifiers and public keys that is used in the confidential communication component to set up random routes silently, page 119

Peer Things. Communication platform developed at Siemens, supporting video communication, voice communication, instant messaging, etc, page 34

Peer-to-peer application. Entire peer-to-peer-based application, including the overlay layer, the service layer, and the application layer, page 20

Peer-to-peer overlay. Addressing scheme that establishes a routing topology independently of the underlying network topology, page 20

Peer-to-peer paradigm. System paradigm with no specialized entities for certain tasks. All entities have the same abilities and operate in the same way, page 20

Power generator. In the decentralized power generation scenario, end users may be power generators at the same time, page 42

Process-driven collaboration paradigm. A business collaboration paradigm emphasizing long-term processes with a rarely changing set of partners, page 35

Public key infrastructure (PKI). Provides authentication, encryption, and data integrity, page 55

Query expression. Path for addressing data in the routing and aggregation infrastructure presented in Section 6. Query expressions can be singlecast, multicast, or broadcast, page 97

Random oracle model. A hash function operates in the random oracle model, if it yields independent uniformly distributed random variables, even if the values to be hashed differ only in a single Bit, page 83

Rationalist. See constructivistic, page 7

Remoting handler. Internal API within the RMF, allows to implement hooks handling outgoing messages, page 119

Rendez-vous. Initially known peer that helps new peers to bootstrap the routing tables when they join the overlay, page 155

Replication group. The group of peers that keep copies of a resource is called the replication group for that resource, page 50

Request for quotations (RfQ). Business document sent from the OEM to its suppliers, asking them propose technical specifications, page 36

Resource management framework (RMF). A DHT-based publish/subscribe infrastructure developed at Siemens Corporate Technology, page 116

Routing. According to the architecture presented in Section 3, routing and stabilization are the only two functionalities implemented on the overlay layer. Routing means resolving peers that are responsible for a set of identifiers, page 29

Self-healing. If a peer in a replication group fails, a new peer becomes part of that group and copies of the resources are stored on that new peer. Therefore, replication is self-healing, page 129

Service layer. Component-based middleware implementing customized, application-specific APIs on top of a generic, lightweight overlay, page 28

Service-oriented architecture (SOA). Architectural style for distributed systems, where complex processes are decomposed into activities, page 13

Service. A mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description [25], page 13

Set of identifiers. A set of identifiers can be queried in a DHT, and the DHT resolves the set of peers being responsible for this set of identifiers. Most DHTs restrict the set of identifiers to have only one element, page 29

Set of peers. A set of identifiers can be queried in a DHT, and the DHT resolves the set of peers being responsible for this set of identifiers. Most DHTs resolve only one responsible peer per query, page 29

Shadow node. In the routing and aggregation infrastructure in Section 6, shadow nodes are used for routing response messages, page 102

Simp. A simple, single-threaded, message-level overlay simulator, page 111

Simulation. The most common way of validating software systems. However, direct experiments and analytical evaluation should be preferred over simulation, page 108

Skip list. Data structure in the EPHT, improving efficiency of range queries in sparsely populated overlays, page 79

Small and medium enterprises (SMEs). In the CPD scenario presented in Section 4, the SMEs are companies that do not have an own IT department for managing servers. Therefore, SMEs benefit from peer-to-peer-based plug-and-play infrastructures, page 36

Space filling curve. In an n-dimensional unit cube, a space filling curve is a continuous path that passes every point in the cube once, page 88

Stabilization. According to the architecture presented in Section 3, routing and stabilization are the only two functionalities implemented on the overlay layer. Stabilization keeps the routing table updated if peers join or leave the overlay, page 29

Structured peer-to-peer overlay. An overlay topology where each peer is responsible for a certain range of identifiers, page 22

Subscription resource. Used to implement a simple publish/subscribe mechanism. The subscription resource is stored on the peer being responsible for the original resource, and it contains the list of all subscribers, page 60

Super peer. Peer that provides some more services than regular peers, e.g. an additional search index, page 22

Tapestry. A DHT developed at the University of California at Berkeley, page 157

Tightly coupled. Two components are tightly coupled, if they have strong mutual impact on each other, page 147

Tomcat. A Java Servlet implementation and Web Server. The AXIS SOAP engine runs as a Servlet within Tomcat, page 117

Unstructured peer-to-peer overlay. An overlay topology where each peer just knows some arbitrary set of neighboring peers. In unstructured overlays, queries are resolved by flooding them recursively to all neighbors, page 21

Utility. In the distributed power generation scenario in this thesis, a utility is a company selling electrical power to end users, and buying electrical power from end users, page 42

Virtual peers. A load balancing technique that was introduced together with the Chord DHT, page 61

Virtual private network (VPN). Network layer on top of TCP/IP, that is only accessible to authenticated members, and that provides encryption, page 70

Wildcard search. Synonym for range query, page 80

Bibliography

[1] Christina L. Abad, William Yurcik, and Roy H. Campbell. A survey and comparison of end-system overlay multicast solutions suitable for network-centric warfare. In Raja Suresh, editor, *Battlespace Digitization and Network-Centric Systems IV*, volume 5441 of *SPIE*, pages 215--226. SPIE, 2004.

[2] Serge Abiteboul, Omar Benjelloun, Bogdan Cautis, Ioana Manolescu, Tova Milo, and Nicoleta Preda. Lazy query evaluation for active xml. In *SIGMOD '04: Proc. of the 2004 ACM SIGMOD international conference on Management of data*, pages 227--238, New York, NY, USA, 2004. ACM Press.

[3] Serge Abiteboul, Omar Benjelloun, and Tova Milo. The active xml project: an overview. Technical Report 331, Gemo, INRIA-Futurs, 10 2005.

[4] Serge Abiteboul, Ioana Manolescu, and Nicoleta Preda. Constructing and querying peer-to-peer warehouses of xml resources. In *ICDE '05: Proc. of the 21st International Conference on Data Engineering*, pages 1122--1123, Washington, DC, USA, 2005. IEEE Computer Society.

[5] Carlisle Adams and Steve Lloyd. *Understanding Public-Key Infrastructures: Concepts, Standards, and Deployment Considerations*. O'Reilly, 1999.

[6] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home: An experiment in public-resource computing. *Communications of the ACM*, 45(11):56--61, 2002.

[7] James Aspnes and Gauri Shah. Skip graphs. In *SODA '03: Proc. of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 384--393, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[8] Jerry Banks and Randall R. Gibson. Don't simulate when: 10 rules for determining when simulation is not appropriate. *IIE Solutions*, 29(9):30--32, September 1997.

[9] Jerry Banks, John S. Carson II, Barry L. Nelson, and David Nicol. *Discrete-event System Simulation*. Prentice Hall, 2001.

[10] Udo Bartlang, Fabian Stäber, and Jörg P. Müller. Introducing a jsr-170 standard-compliant peer-to-peer content repository to support business collaboration. In *Proc. of eChallenges 2007*, volume 4 of *Exploiting the Knowledge Economy: Issues, Applications and Case Studies*, pages 814--821, Amsterdam, Berlin, Oxford, Tokyo, Washington, DC, USA, 2007. IOS Press.

[11] Len Bass, Paul Clements, and Rick Kazmann. *Software Architecture in Practice*. Addison-Wesley Professional, 2nd edition, 2003.

[12] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93: Proc. of the 1st ACM conference on Computer and communications security*, pages 62--73, New York, NY, USA, 1993. ACM Press.

[13] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: Supporting scalable multi-attribute range queries. In *SIGCOMM '04: Proc. of the SIGCOMM Symposium on Communications Architectures and Protocols*, pages 353--366, Portland, OR, USA, 2004.

[14] N. Bieberstein, S. Bose, L. Walker, and A. Lynch. Impact of service-oriented architecture on enterprise systems, organizational structures, and individuals. *IBM Systems Journal*, 44(4):691--708, 2005.

[15] Markus Böhm. Requirements zur PeerThings Skalierbarkeit, 2005. Internal memo at Siemens Communications, Fixed Networks Group.

[16] Angela Bonifati, Ugo Matrangolo, Alfredo Cuzzocrea, and Mayank Jain. Xpath lookup queries in p2p networks. In Alberto H. F. Laender, Dongown Lee, and Marc Ronthaler, editors, *WIDM '04: Proc. of the 6th ACM International Workshop on Web Information and Data Management*, pages 48--55. ACM Press, 2004.

[17] R. Braden. Requirements for internet hosts -- communication layers. RFC 1122 (Standard), October 1989. Updated by RFCs 1349, 4379.

[18] Encyclopaedia Britannica. *Encyclopaedia Britannica*. 15th edition.

[19] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. One ring to rule them all: Service discovery and binding structured peer-to-peer overlay networks. In *SIGOPS '02: Proc. of the 10th ACM*

SIGOPS European Workshop, pages 140--145, New York, NY, USA, 2002. ACM Press.

[20] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. Scribe. a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications (JSAC)*, 20(8):1489--1499, 2002.

[21] L. Cherbakov, G. Galambos, R. Harishankar, S. Kalyana, and G. Rackham. Impact of service orientation at the business level. *IBM Systems Journal*, 44(4):653--668, 2005.

[22] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In Hannes Federrath, editor, *Designing Privacy Enhancing Technologies. Proc. of the International Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pages 46--66. Springer, 2001.

[23] Bram Cohen. Incentives build robustness in bittorrent, 2003. Workshop on Exonomics of Peer-to-Peer Systems. Published on the workshop's website, http://www.sims.berkeley.edu/p2pecon.

[24] European Commission. Information technology security evaluation criteria (itsec), 1991. Harmonised Criteria of France, Germany, the Netherlands, and the United Kingdom.

[25] OASIS Consortium. Reference model for service oriented architecture 1.0, 2006. document identifier: soa-rm-cs.

[26] Russ Cox, Frank Dabek, Frans Kaashoek, Jinyang Li, and Robert Morris. Practical, distributed network coordinates. *SIGCOMM Computer Communication Review*, 34(1):113--118, 2004.

[27] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *SOSP '01: Proc. of the eighteenth ACM symposium on Operating systems principles*, pages 202--215. ACM Press, 2001.

[28] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a common API for structured peer-to-peer overlays. In *IPTPS'03: Peer-to-Peer Systems II, Second International Workshop*, volume 2735 of *Lecture Notes in Computer Science*, pages 33--44, Berlin, Heidelberg, 2003. Springer.

[29] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type iii anonymous remailer protocol. In *Proc. of the 2003 IEEE Symposium on Security and Privacy*, pages 2--15, 2003. http://mixminion.net.

[30] Luka Divac-Krnic and Ralf Ackermann. Security-related issues in peer-to-peer networks. In *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*, pages 529--545. Springer, 2005.

[31] John R. Douceur. The sybil attack. In Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen, editors, *IPTPS '02: Proc. of the 1st International Workshop on Peer-to-Peer Systems. Revised Papers*, volume 2429/2002 of *Lecture Notes in Computer Science*, pages 251--260. Springer, 2002.

[32] Schahram Dustar, Harald Gall, and Manfred Hauswirth. *Software-Architekturen*. Springer, Berlin, Heidelberg, 2003.

[33] Klaus Fischer, Jörg P. Müller, Fabian Stäber, and Thomas Friese. Using peer-to-peer protocols to enable implicit communication in a BDI agent architecture. In *ProMAS'06: Proc. of the 4th International Workshop on Programming Multi-Agent Systems*, volume 4411 of *Lecture Notes in Artificial Intelligence*, pages 15--37, Berlin, Heidelberg, 2006. Springer.

[34] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-peer communication across network address translators. In *ATEC'05: Proc. of the USENIX Annual Technical Conference 2005 on USENIX Annual Technical Conference*, pages 13--13, Berkeley, CA, USA, 2005. USENIX Association.

[35] Ian Foster. Head node -- the grid. *Cluster World*, 1(1):01--02, 2004.

[36] Ian Foster. Globus toolkit version 4: Software for service-oriented systems. In *NPC'05: Proc. of the IFIP International Conference on Network and Parallel Computing*, volume 3779 of *Lecture Notes in Computer Science*, pages 2--13, Berlin, Heidelberg, 2005. Springer.

[37] Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[38] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The physiology of the grid, an open grid services architecture for distributed systems integration, 2002. Presented at the GGF5: The 5th Global Grid Forum.

[39] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organization. *International Journal of Supercomputer Applications and High Performance Computing*, 15(3):200--222, 2001.

[40] Ian T. Foster and Adriana Iamnitchi. On death, taxes, and the convergence of peer-to-peer and grid computing. In *IPTPS'03: Proc. of the 2nd International Workshop on Peer-to-Peer Systems*, Berkeley, CA, USA, 2003. Springer.

[41] Michael J. Freedman, Eric Freudenthal, and David Mazières. Democratizing content publication with coral. In *NSDI'04: Proc. of the 1st Conference on Symposium on Networked Systems Design and Implementation*, pages 18--18, Berkeley, CA, USA, 2004. USENIX Association.

[42] Thomas Friese, Jörg P. Müller, Matthew Smith, and Bernd Freisleben. A robust business resource management framework based on a peer-to-peer infrastructure. In *CEC '05: Proc. of the 7th International IEEE Conference on E-Commerce Technology*, pages 215--222. IEEE Press, 2005.

[43] Li Gong. Jxta: A network programming environment. *IEEE Internet Computing*, 5(3):88--95, May/June 2001.

[44] Vladimir Gorodetsky, Oleg Karsaev, Vladimir Samoylov, and Sergey Serebryakov. P2p agent platform: Implementation and testing. In *AAMAS'07: Proc. of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2007.

[45] The Open Group. Definition of SOA, 2006. Version 1.1.

[46] Saikat Guha and Paul Francis. Characterization and measurement of tcp traversal through nats and firewalls. In *IMC'05: Proc. of the Internet Measurement Conference 2005 on Internet Measurement Conference*, pages 18--18, Berkeley, CA, USA, 2005. USENIX Association.

[47] Qi He, Mostafa H. Ammar, George F. Riley, Himanshu Raj, and Richard Fujimoto. Mapping peer behavior to packet-level details: A framework for packet-level simulation of peer-to-peer systems. In *MASCOTS'03: 11th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*, pages 71--78. IEEE Computer Society, 2003.

[48] Xiaojun Hei, Chao Liang, Jian Liang, Yong Liu, and Keith W. Ross. Insights into PPLive: A measurement study of a large-scale p2p iptv system, 2006.

Presented at the *IPTV Workshop* at the *WWW'06: 15th International World Wide Web Conference*.

[49] Peter Hessheimer. Konzeption und prototypische umsetzung eines verteilten und dynamischen simulators für peer-to-peer systeme, 2007.

[50] Tobias Hoßfeld, Andreas Binzenhöfer, Daniel Schlosser, Kolja Eger, Jens Oberender, Ivan Dedinski, and Gerald Kunzmann. Towards efficient simulation of large scale p2p networks. Technical Report 371, University of Würzburg, Institute of Computer Science, Würzburg, Germany, 10 2005.

[51] Ryan Huebsch, Brent Chun, Joseph Hellerstein, Boon Thau Loo, Petros Maniatis, Timothy Roscoe, Scott Shenker, Ion Stoica, and Aydan R. Yumerefendi. The architecture of PIER: an internet-scale query processor. In *CIDR '05: Proc. of the 2nd Conference on Innovative Data Systems Research*, pages 28--43, 2005.

[52] Piotr Indyk, Rajeev Motwani, Prabhakar Raghavan, and Santosh Vempala. Locality-preserving hashing in multidimensional spaces. In *STOC '97: Proc. of the twenty-ninth annual ACM symposium on Theory of computing*, pages 618--625, New York, NY, USA, 1997. ACM Press.

[53] Nicholas R. Jennings, Katia Sycara, and Michael Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7--38, 1998.

[54] Brent ByungHoon Kang. *S2D2: A Framework for Scalable and Secure Optimistic Replication*. PhD thesis, UC Berkeley, 2004.

[55] Gerald Kunzmann. Abschlussbericht: Forschung und Entwicklung von effizienten und skalierbaren Suchverfahren für verteilte Systeme. Technical report, Technische Universität München, 2007.

[56] Jim Kurose and Keith Ross. *Computer Networking, A Top-Down Approach Featuring the Internet*. Addison Wesley Longman, Inc., 2001.

[57] Leslie Lamport. Paxos made simple. *SIGACT News*, 32(4):18--25, 2001.

[58] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *TOPLAS: ACM Transactions on Programming Languages and Systems*, 4(3):382--401, 1982.

[59] Matthew Leslie, Jim Davies, and Todd Huffman. Replication strategies for reliable decentralised storage. In *ARES '06: Proc. of the First International Conference on Availability, Reliability and Security*, pages 740--747, 2006.

[60] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7(1):72--93, 2005.

[61] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. *SIGMETRICS Perform. Eval. Rev.*, 30(1):258--259, 2002.

[62] Peter Mahlmann and Christian Schindelhauer. *Peer-to-Peer Netzwerke*. Springer, Berlin, Heidelberg, 2007.

[63] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '02: Proc. of the 1st International Workshop on Peer-to-Peer Systems*, pages 53--65, London, UK, 2002. Springer.

[64] Brenda M. Michelson. Event-driven architecture overview, 2006. Patricia Seybold Group and Elemental Links, Inc.

[65] Michael Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094--1104, 2001.

[66] Athicha Muthitacharoen, Seth Gilbert, and Robert Morris. Etna: a fault-tolerant algorithm for atomic mutable dht data. Technical Report MIT-LCS-TR-993, MIT Computer Science and Artificial Intelligence Laboratory, 2005.

[67] Wolfgang Nejdl, Boris Wolf, Changtau Qu, Stefan Decker, Michael Sintek, Ambjörn Naeve, Mikael Nilsson, Matthias Palmér, and Tore Risch. Edutella: A p2p networking infrastructure based on RDF. In *WWW '02: Proc. of the Eleventh International World Wide Web Conference*, pages 604--615, 2002.

[68] Newport Networks. Lawful intercept overview, 2006. white paper 91-0087-01-0001-A, http://www.newport-networks.com/cust-docs/87-Lawful-Intercept.pdf.

[69] Michael Niebergall. Strategien zur Lastverteilung für Suchanfragen in strukturierten P2P-Overlay-Netzwerken, Juli 2006.

[70] Andy Oram, editor. *Peer-to-Peer, Harnessing the Power of Disruptive Technologies*. O'Reilly, Sebastopol, CA, USA, 1st edition, 2001.

[71] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A blueprint for introducing disruptive technology into the internet. *SIGCOMM Computer Communication Review*, 33(1):59--64, 2003.

[72] C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *SPAA'97: Proc. of the 9th annual ACM symposium on parallel algorithms and architectures*, pages 311--320, New York, NY, USA, 1997. ACM.

[73] ATHENA Project. Architecture of soa platforms, 2005. Deliverable D.A5.3.

[74] ATHENA Project. Handbook nehemiah for bpm, 2005. Annex 1 for Deliverable D.A2.4.

[75] ATHENA Project. Architecture for enactment and integration of cross-organizational business processes, 2006. Deliverable D.A2.4.

[76] ATHENA Project. Brmf -- peer-to-peer business resource management framework, 2006. Annex 2 for Deliverable D.A6.4.

[77] ATHENA Project. Handbook gabriel for bpm, 2006. Annex 1' for Deliverable D.A2.4.

[78] ATHENA Project. Handbook maestro for bpm, 2006. Work Package A2.2.

[79] ATHENA Project. Model-driven and adaptable interoperability infrastructure, 2006. Deliverable D.A6.4.

[80] ATHENA Project. Specification of a cross-organisational business process model, 2006. Deliverable D.A2.2.

[81] ATHENA Project. Piloting including technology testing coordination and pilot infrastructure, 2007. Deliverable D.B5.4.

[82] Sriram Ramabhadran, Sylvia Ratnasamy, Joseph M. Hellerstein, and Scott Shenker. Prefix hash tree -- an indexing data structure over distributed hash tables. announced at the 23rd annual ACM symposium on Principles of distributed computing, PODC '04.

[83] Venugopalan Ramasubramanian and Emin Gün Sirer. The design and implementation of a next generation name service for the internet. In *SIGCOMM '04: Proc. of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 331--342, New York, NY, USA, 2004. ACM Press.

[84] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proc. of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161--172, New York, NY, USA, 2001. ACM Press.

[85] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *TISSEC: ACM Transactions on Information and System Security*, 1(1):66--92, 1997.

[86] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz. Pond: The oceanstore prototype. In *FAST '03: Proc. of the 2nd USENIX Conference on File and Storage Technologies*, pages 1--14, Berkeley, CA, USA, 2003. USENIX Association.

[87] Sean Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a dht. In *USENIX '04: Proc. of the USENIX Technical Conference*, pages 127--140, Boston, MA, USA, June 2004.

[88] Simon Rieche, Leo Petrak, and Klaus Wehrle. A thermal-dissipation-based approach for balancing data load in distributed hash tables. In *LCN '04: Proc. of the 29th Annual IEEE International Conference on Local Computer Networks*, pages 15--23, 2004.

[89] George F. Riley, Richard M. Fujimoto, and Mostafa H. Ammar. A generic framework for parallelization of network simulations. In *MASCOTS'99: Proc. of the 7th Int. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 128--135, 1999.

[90] John Risson and Tim Moors. Survey and research towards robust peer-to-peer networks: Search methods. *Computer Networks*, 50(17):3485--3521, 2006.

[91] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Middleware '01, Proc. of the IFIP/ACM International Conference on Distributed Systems Platforms*, volume 2218 of *Lecture Notes in Computer Science*, pages 329--350, 2001.

[92] Steffen Rusitschka, Markus Böhm, and Uwe Leutritz. Peer Things, 2005. Internal project report at Siemens Communications, Fixed Networks Group.

[93] Steffen Rusitschka and Alan Southall. The resource management framework: A system for managing metadata in decentralized networks using peer-to-peer technology. In *Agents and Peer-to-Peer Computing*, volume 2530 of *Lecture Notes in Computer Science*, pages 144--149. Springer, 2003.

[94] Christina Schmidt and Manish Parashar. Enabling flexible queries with guarantees in P2P systems. *IEEE Internet Computing*, 8(3):19--26, 2004.

[95] Bruce Schneier. *Applied Cryptography*. John Wiley & Sons, 2nd edition, 1996.

[96] Clip2 Distributed Search Services. The gnutella protocol specification. v0.4.

[97] Adam Slagell, Rafael Bonilla, and William Yurcik. A survey of pki components and scalability issues. In *WIA'06: Proc. of the 25th IEEE Int. Performance Computing and Communications Conference, IPCCC 2006*, pages ??--??, 2006.

[98] Mudhakar Srivatsa and Ling Liu. Vulnerabilities and security threats in structured overlay networks: a quantitative analysis. In *ACSAC '04: Proc. of the 20th Annual Computer Security Applications Conference*, pages 252--261. IEEE Press, 2004.

[99] Fabian Stäber, Udo Bartlang, and Jörg P. Müller. Using onion routing to secure peer-to-peer supported business collaboration. In *Proc. of eChallenges 2006*, volume 3 of *Exploiting the Knowledge Economy: Issues, Applications and Case Studies*, pages 181--188, Amsterdam, Berlin, Oxford, Tokyo, Washington, DC, USA, 2006. IOS Press.

[100] Fabian Stäber, Christoph Gerdes, and Jörg P. Müller. A peer-to-peer-based service infrastructure for distributed power generation. In *IFAC WC'08: Proc. of the 13th International Federation of Automatic Control World Congress*, 2008. to appear.

[101] Fabian Stäber, Gerald Kunzmann, and Jörg P. Müller. Extended prefix hash trees for a distributed phone book application. In *ICPADS'07: Proc. of the 17th International Conference on Parallel and Distributed Systems*. IEEE Computer Society Press, 2007.

[102] Fabian Stäber and Jörg P. Müller. Evaluating peer-to-peer for loosely coupled business collaboration: A case study. In *BPM'07: Proc. of the 5th International Conference on Business Process Management*, volume 4714

of *Lecture Notes in Computer Science*, pages 141--148, Berlin, Heidelberg, 2007. Springer.

[103] Fabian Stäber, Giorgio Sobrito, Jörg P. Müller, Udo Bartlang, and Thomas Friese. Interoperability challenges and solutions in automotive collaborative product development. In *I-ESA'07: Proc. of the 3rd International Conference on Interoperability for Enterprise Software and Applications*, volume 2 of *Enterprise Interoperability*, pages 709--720, London, UK, 2007. Springer.

[104] Ralf Steinmetz and Klaus Wehrle, editors. *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 2005.

[105] Ralf Steinmetz and Klaus Wehrle. *What is Peer-to-Peer About?*, volume 3485 of *Lecture Notes in Computer Science*, pages 9--16. Springer, Berlin, Heidelberg, 2005.

[106] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proc. of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149--160, San Diego, CA, USA, 2001. ACM Press.

[107] Daniel Stutzbach and Reza Rejaie. Understanding churn in peer-to-peer networks. In *IMC '06: Proc. of the 6th ACM SIGCOMM on Internet Measurement*, pages 189--202, New York, NY, USA, 2006. ACM Press.

[108] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall International, Upper Saddle River, NJ, USA, 2003.

[109] TopWare CD-Service GmbH. D-Info 97 CDROM. http://www.d-info.de.

[110] International Telecommunication Union. ASN.1 encoding rules -- specification of basic encoding rules (BER), canonical encoding rules (CER) and distinguished encoding rules (DER). Recommendation X.690, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, 12 1997.

[111] International Telecommunication Union. Information technology -- open systems interconnection -- the directory: Public-key and attribute certificate frameworks. Recommendation X.509, Telecommunication Standardization Sector of ITU, Geneva, Switzerland, 3 2000.

[112] Michael Valocchi, Allan Schurr, John Juliano, and Ekov Nelson. Plugging in the consumer: Innovating utility business models for the future. Technical report, IBM Institute for Business Value study, November 2007.

[113] Mathias Weske. *Business Process Management. Concepts, Languages, Architectures*. Springer, Berlin, Heidelberg, 2007.

[114] IEEE FIPA P2PNA WG. Functional architecture specification, 2006. Draft 0.12, http://www.fipa.org/subgroups/P2PNA-WG-docs/P2PNA-Spec-Draft0.12.doc.

[115] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115--152, 1995.

[116] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41--53, 2004.

[117] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California at Berkeley, 2001.

[118] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV'01: Proc. of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 11--20. ACM Press, 2001.

[119] Stefan Zoels, Simon Schubert, Wolfgang Kellerer, and Zoran Despotovic. Hybrid dht design for mobile environments. In *AP2PC'06: Proc. of 5th International Workshop on Agents and Peer-to-Peer Computing*, LNCS. Springer Verlag, 2006.